Napredni razvoj programske potpore za web

- predavanja -

2022./2023.

---

10. Atributi kvalitete web-aplikacija (i kako ih poboljšati…)

Creative Commons

- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- pod sljedećim uvjetima:
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

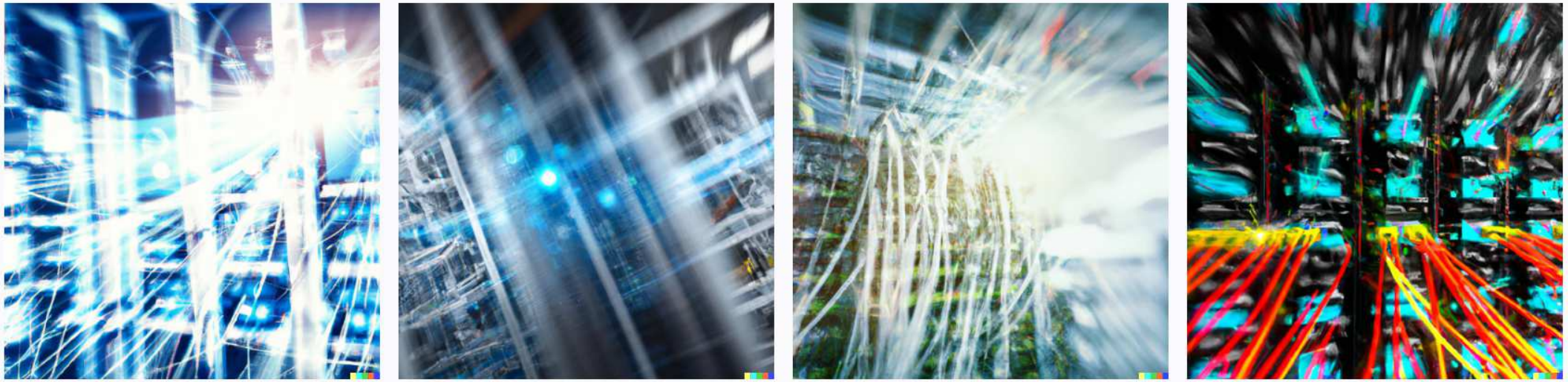
of visits are abandoned if a mobile site takes longer than 3 seconds to load.

Google Data, Global, n=3,700 aggregated, anonymized Google Analytics data from a sample of mWeb sites opted into sharing benchmark data, March 2016.

https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/

Zahtjevi na sustav

1. **Svojstva sustava, funkcionalni** zahtjevi (*system features – functional reqs*)
   - Definiraju funkcionalnost (nekog dijela) sustava, apstrahirano output = f(input)
   - Npr. sustava mora omogućiti pretraživanje hrane, odabir hrane (dodavanje u košaricu), predavanja i plaćanje narudžbe…
   - Koristimo: use-cases, user-flows…
   - Općenito: bilo koja arhitektura može ostvariti bilo koja funkc. svojstva ☹
2. **Atributi kvalitete, nefunkcionalni** zahtjevi (*quality attributes, non-functional reqs*)
   - Svojstva koja sustava treba imati, npr. dostupnost, skalabilnost, pouzdanost, sigurnost, …
   - Jako utječu na arhitekturu sustava
3. **Zadana ograničenja** sustava (*system constraints*)
   - Tehnička, poslovna, pravna
   - Npr. ograničen budžet, ograničen broj i znanje djelatnika, licence, vremenski rokovi, itd.



U nastavku se bavimo samo s:

2. nefunkcionalni zahtjevi

"Quality is never an accident;
it is always the result of high intention, sincere effort, intelligent direction and skillful execution;
it represents the wise choice of many alternatives."

William A. Foster

Atributi kvalitete - motivacija

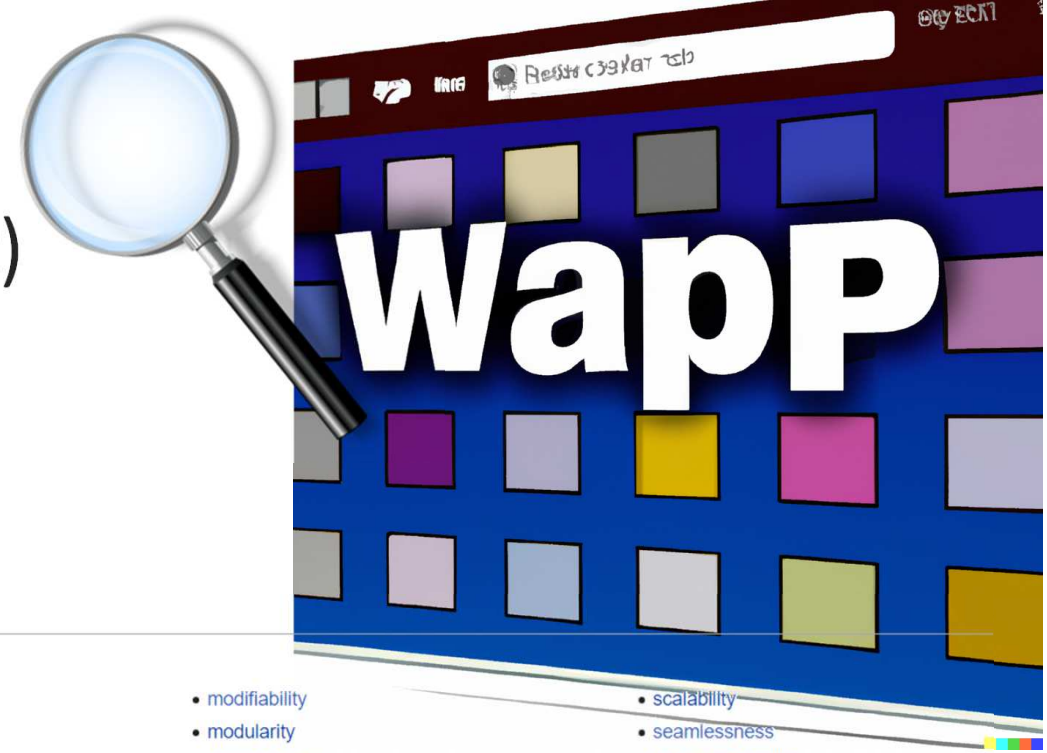
- Koji su razlozi za redizajn/prepravak sustava?
  - Sustav je **spor**
  - Nije **skalabilan**
  - Težak za **održavanje**
  - Ima **sigurnosne** propuste
  - Spor **razvoj**
  - **Preskup**
  - …
- Redizajnom sustava ne mijenjamo funkcionalnost
  - Neke od atributa korisnici osjećaju (npr. brz/spor), a neke ne (teškoće razvoja i održavanja)

Atributi kvalitete

- Atribut kvalitete je **mjerljivo** ili **provjerljivo** svojstvo sustava koje govori koliko dobro sustav **zadovoljava** potrebe s obzirom na neku **dimenziju kvalitete.**
- Izravno utječe na arhitekturu sustava (i obratno)
- Na primjer:
  - Sustav mora vratiti odgovor u tražilici za manje od 2s
  - Sustav mora biti raspoloživ barem 99.9% vremena
- „Mjerljivo ili provjerljivo”:
  - Sustav mora vratiti odgovor unutar 2s
  - ~~Sustav mora brzo vratiti odgovor~~
- Neki atributi kvalitete su u suprotnosti s drugima (npr. sigurnost je tipično u suprotnosti sa „svima”), tako da ne možemo maksimizirati po svima
  - Tradeoff ☹

Atributi kvalitete

1. *Performance* (učinkovitost)
2. *Scalability* (skalabilnost)
3. *Availability* (dostupnost)
4. ...



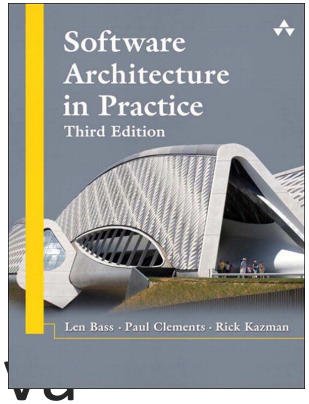
Quality attributes [edit]

Notable quality attributes include:

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- confidentiality
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- redundancy
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

1. Preformance / response time



- *"It's about time!"*
- *Response time (end-to-end latency)* - vrijeme odziva
- **Vrijeme potrebno da dobijemo odgovor**

RT = *waiting time (latency) + processing time*

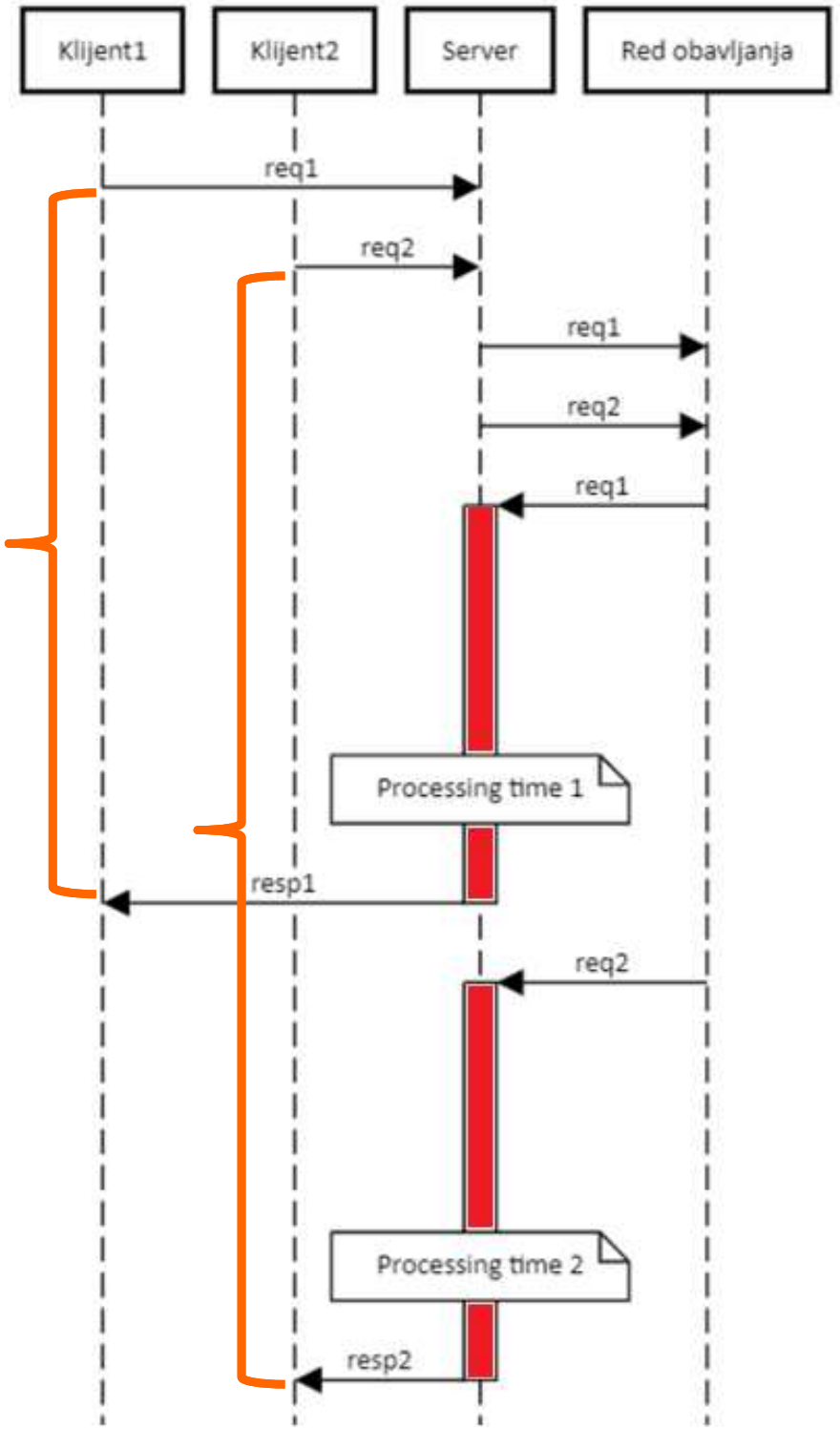
  - **WT** - "putovanje" kroz komunikacijske kanale, čekanje u redovima, itd.
  - **PT** - vrijeme provedeno u "našem kodu" na serveru

- RT se nekad naziva i *End-to-end latency*

Paziti kod mjerenja - izmjeriti oboje!

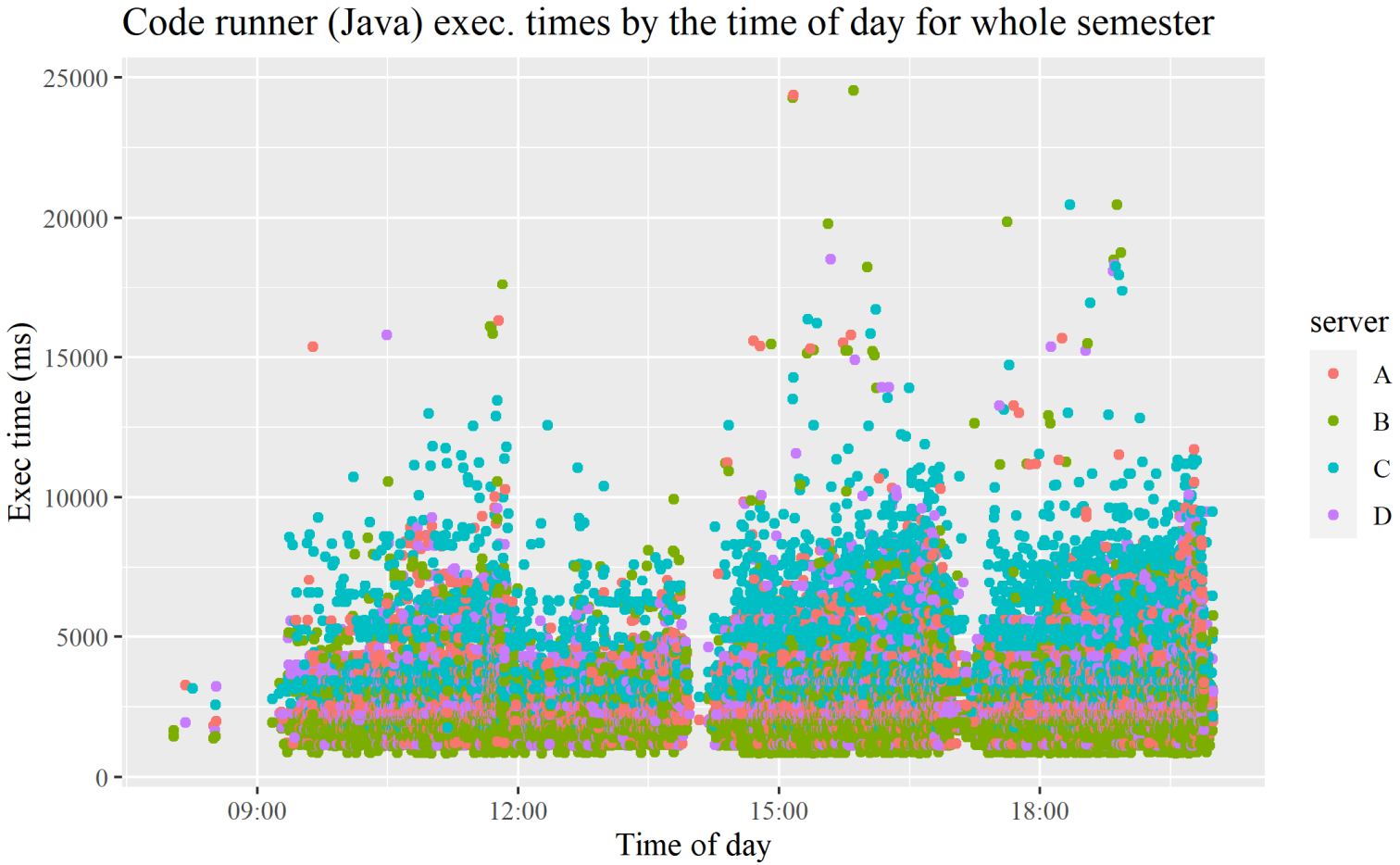
- Klasična pogreška: mjeriti samo processing time na serveru, tada:
  - (neka je PT1 = PT2 = 2s)
  - AVG("RT") = 2.0s

- Treba mjeriti iz perspektive klijenta, tada:
  - RT1 = 2s + 2s = 4s
  - RT2 = 3.5s + 2s = 5.5s
  - AVG(RT) = 4.75s



RT -> koje mjere koristiti?

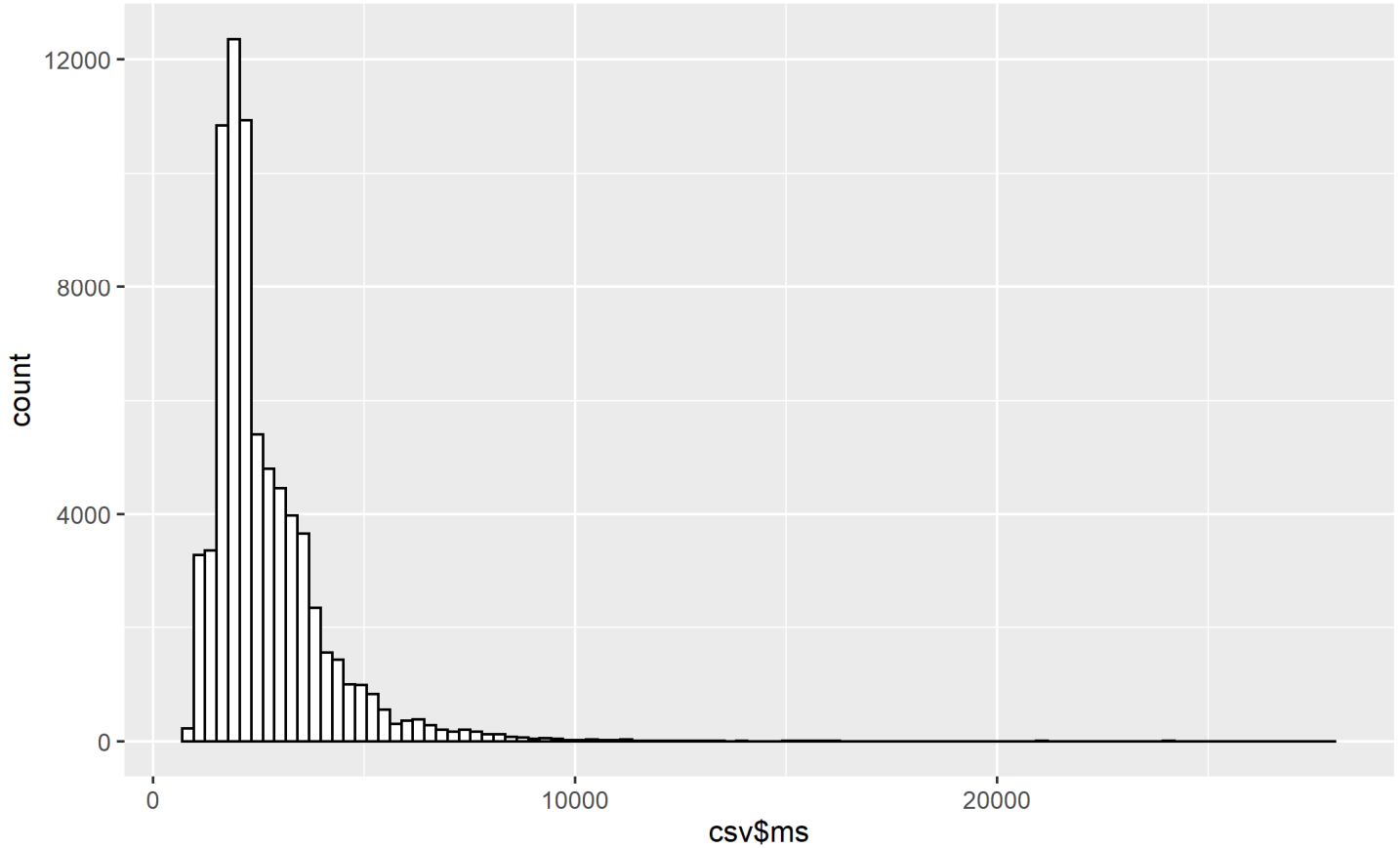
- Primjer, RT s tri poslužitelja:



- Medijan 2262.6ms
- Prosjek 2702.9ms

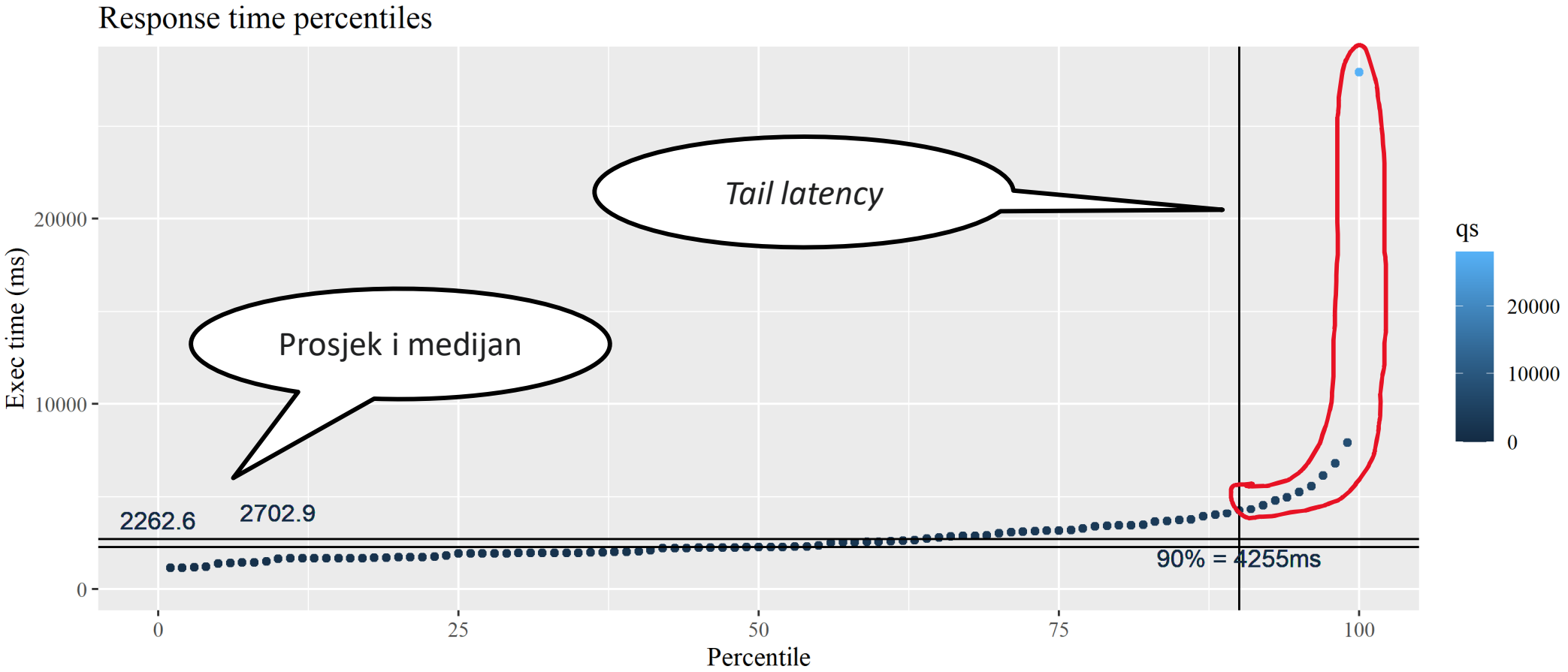
dobra metrika?

RT -> histogram (distribucija RT-a)



- N-ti **percentil** nekog skupa podataka je vrijednost koja je veća od n% vrijednost iz skupa
  - 50. percentil = median
  - 25. percentil = prvi kvartal, itd.
- Pretvorimo sad histogram u **percentile** –> sljedeći slajd

Percentili *i Tail latency* (1)



- TL je mali postotak RT-a koji traje najdulje (u usporedbi s većinom):
  - Svakako > max(prosjek, medijan)
  - Npr. „zadnjih 10%”

Percentili *i Tail latency* (2)

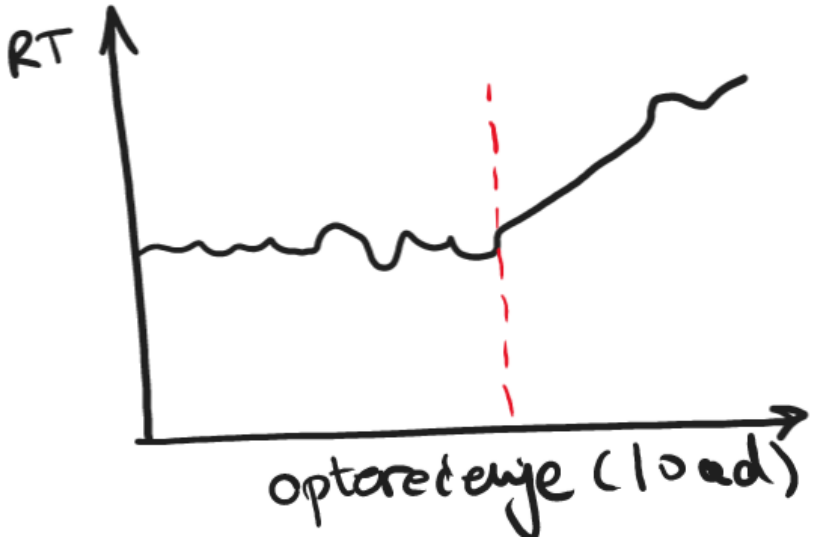
- Idealno - što kraći „rep”
- Ciljeve si onda možemo bolje postaviti koristeći percentile (a ne prosjek i medijan), npr.:
  - RT < 1000ms na 95. percentilu
  - RT < 1500 na 99. percentilu
  - Itd.

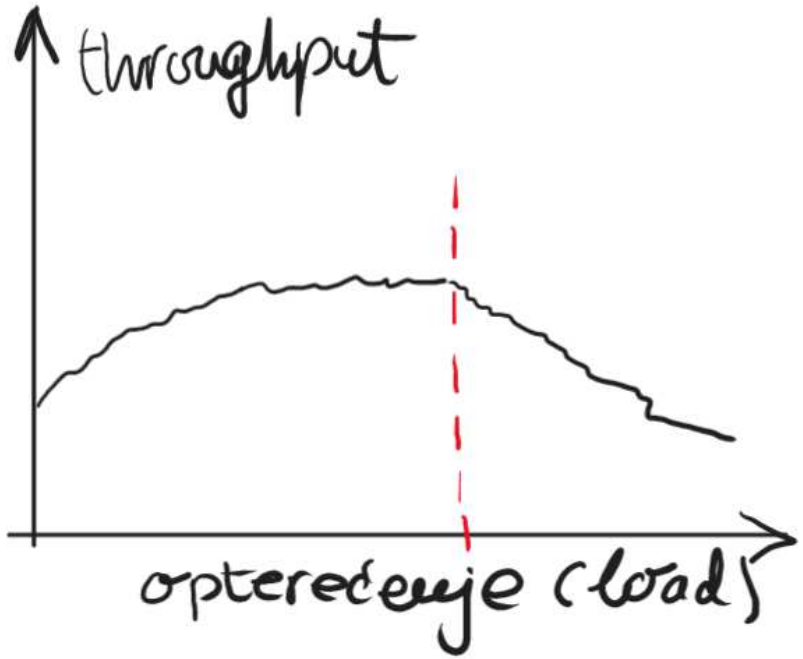
1. Preformance / Throughput - propusnost

- Podjednako važna metrika
- Dva pogleda/definicije:

1. Broj poslova koji se mogu obaviti po jedinici vremena (mjereno u broj/s)
2. Količina podataka koja se može procesirati/ jedinici vremena (mjereno u {k|M|G|…}B/s)

Degradacija performansi?





- „Nešto” je usko grlo:
  - CPU?
  - Memorija?
  - Previše konekcija?
  - …

2. Skalabilnost (1)

- Promet (~opterećenje) nikad nije jednolik
  - Dnevne fluktuacije
  - Sezonske fluktuacije
  - …
  - Organski rast
- Analogija: autoput

Neke definicije:

- Skalabilnost je svojstvo sustava da se nosi s rastućom količinom posla dodavanjem resursa sustavu.

Bondi, André B.

  - Skalabilnost opisuje sposobnost rukovanja većim brojem korisnika, klijenata, podataka, transakcija ili zahtjeva bez utjecaja na korisničko iskustvo
  - Učinkovito (*efficient*) ili ekonomično (*cost-efficient, cost-effective*) ispunjavanje zahtjeva

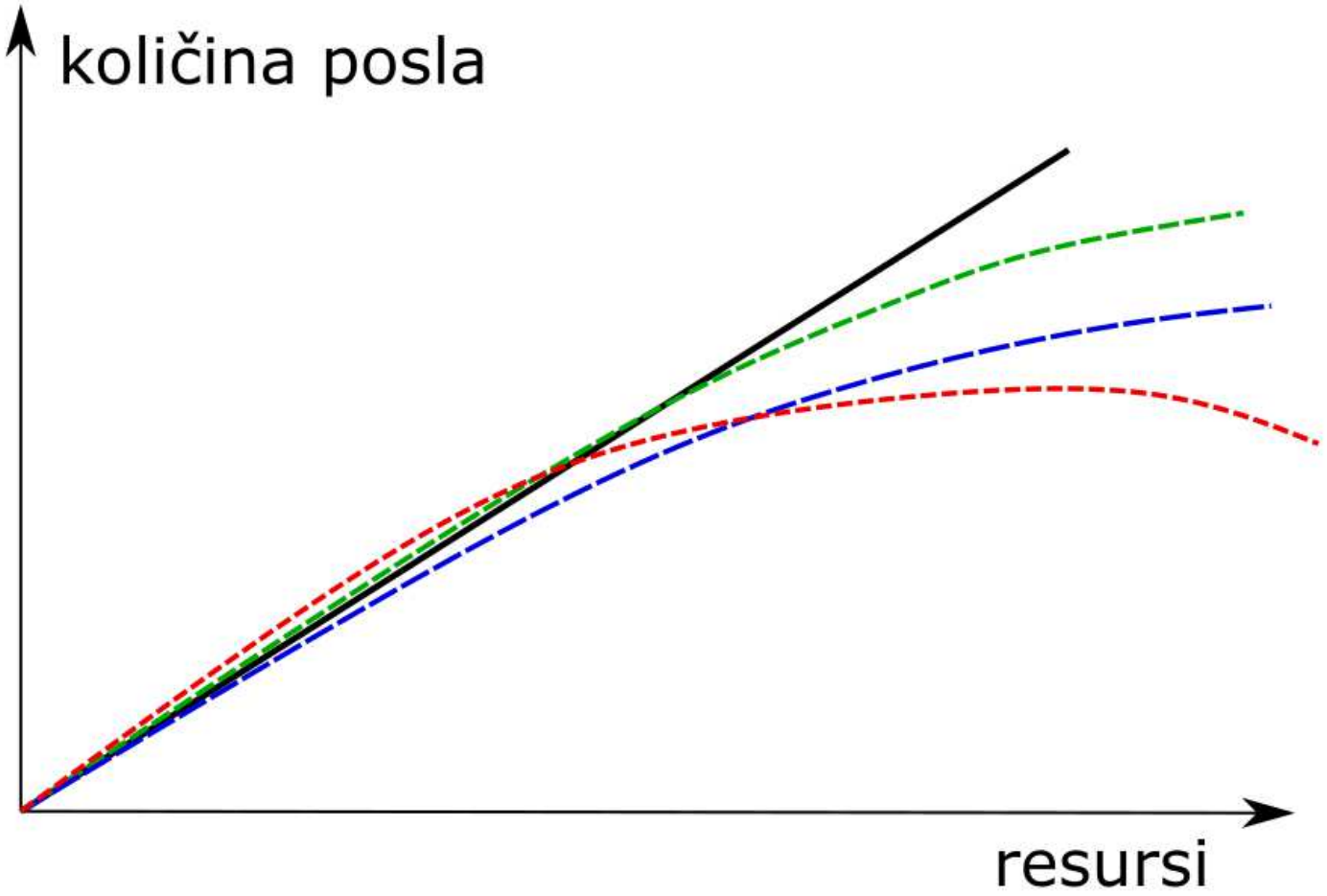
Još neke definicije skalabilnosti… (2)

- *Capacity of the system to handle increasing amount work without affecting existing system.*
- *Scalability is a characteristic of a system, model or function that describes its capability to cope and perform under an increased or expanding workload. A system that scales well will be able to maintain or even increase its level of performance or efficiency when tested by larger operational demands.*
- *The measure of a system's **ability to increase or decrease** in performance and cost in response to changes in application and system processing demands.*
- *Property of software and hardware systems that can improve functionality, performance, and speed by adding or removing more processors, memory, and other resources.*

https://www.igi-global.com/dictionary

2. Skalabilnost (3)

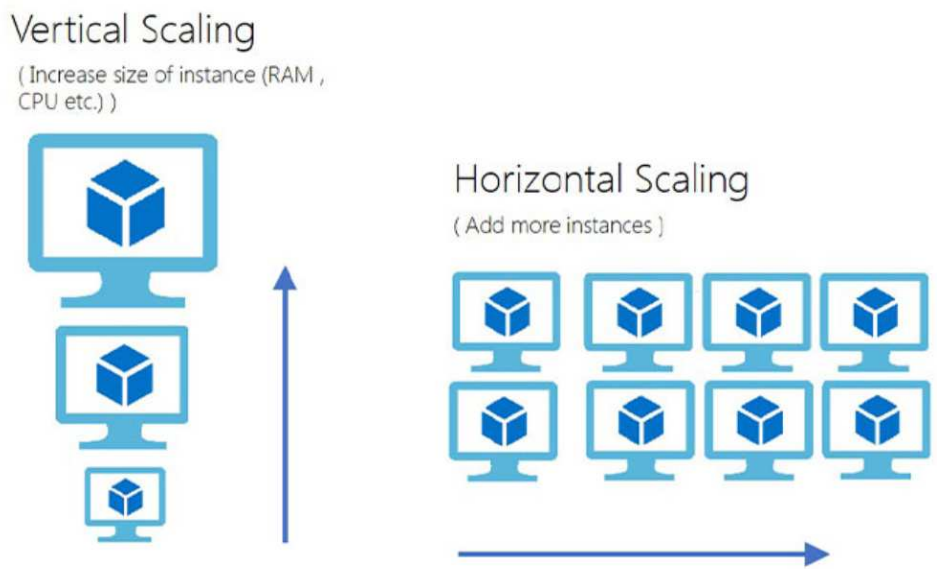
- Idealno (~nemoguće) linearno:



2. Skalabilnost (4)

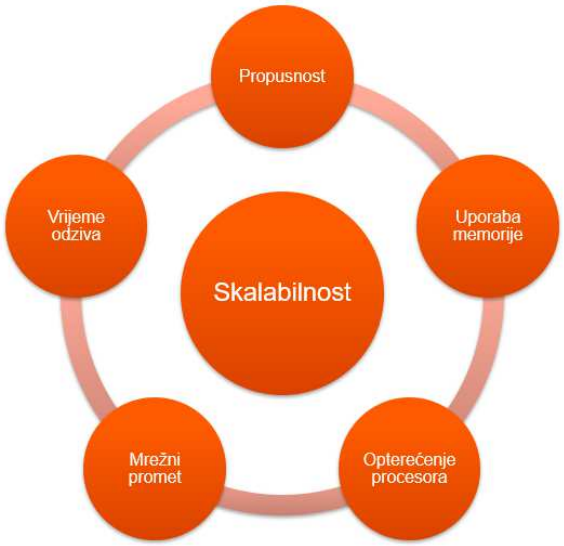
- **Tri vrste** skalabilnosti:

1. **Vertikalna**
   - Dodavanje resursa **jednom računalu**
2. **Horizontalna**
   - Dodavanje novih instanci (resursa) u klaster serverskih računala
3. **Organizacijska**
   - Povećanje produktivnost dodavanjem više radnika



https://www.webairy.com/horizontal-and-vertical-scaling

Značajke važne za ispitivanje skalabilnosti



2. Skalabilnost: vertikalna vs horizontalna (5)

Vertikalna

- ✓ Ekonomično – dodavanje memorije ili procesora, diska na postojeće računalo je jeftinije nego kupovina novog
- ✓ Jednostavnija i brža komunikacija među procesima
- ✓ Jednostavnije održavanje
- ✓ **Ne treba mijenjati kod! Vrijedi za „sve" aplikacije**
- ✓ **Migracija i scale up/down je lagan (posebice u oblaku)**
- X **SPoF**
- X **Veća vjerojatnost prekida u radu**
- X **Ograničenih mogućnosti ☹**

Horizontalna

- ✓ Logički jednostavnije (ne treba detaljno tražiti usko grlo)
- ✓ Manja vjerojatnost prekida rada
- ✓ **Povećana izdržljivost i otpornost na pogreške**
- ✓ **Bolje performanse!**
- ✓ **„neograničenog" rasta**
- X **Promjena koda**
- X **Povećana složenost sustava**
- X Veći inicijalni troškovi

2. Skalabilnost: organizacijska skalabilnost (6)

- Što ako na i sam **razvojni proces** gledamo kroz prizmu skalabilnosti?
  - Rastući posao = razvijanje/programiranje sustava
  - Dodatni resursi = razvojni inženjeri
- Hoćemo li imati linearnu skalabilnost?
  - Veći koordinacijski/organizacijski troškovi
  - Različita znanja
  - *Code merge* konflikti
  - …
  - „sto babica – kilavo dijete”
- Visoko centralizirane arhitekture i centralizirani programski kôd će slabije skalirati
  - Razdvojiti na module/servise/…
  - Manji timovi koji se bave svojim poslom – *separation of concerns*

3. Availability - motivacija

- *This past Tuesday morning Pacific Time an Amazon Web Services engineer was debugging an issue with the billing system for the company's popular cloud storage service S3 and accidentally mistyped a command. What followed was a several hours' long cloud outage that wreaked havoc across the internet and resulted in hundreds of millions of dollars in losses for AWS customers and others who rely on third-party services hosted by AWS.*
- https://www.datacenterknowledge.com/archives/2017/03/02/aws-outage-that-broke-the-internet-caused-by-mistyped-command
- https://awsmaniac.com/aws-outages
- https://github.com/danluu/post-mortems

3. Dostupnost (*Availability*) (1)

- Spremnost sustava da obavlja svoje zadaće
- Jako veliki utjecaj na korisnike!
  - Gubitak novaca i klijenata
- **Dostupnost je vjerojatnost da će sustav biti dostupan (raditi) u nekom proizvoljnom trenutku** (kada se koristi pod navedenim uvjetima u idealnom okruženju)
- ***U**ptime* = vrijeme kad je sustav (bio) dostupan
- ***D**owntime* = vrijeme kad je sustav (bio) nedostupan

$$A = U / (U + D)$$

- Tipično se izražava postotno
  - Ako se *uptime* izrazi postotno onda A = U

3. Dostupnost (*Availability*) (2)

- Alternativna i slična definicija koja je zgodna kad imamo puno različitog hardvera i heterogene sustave (cloud, …)

https://en.wikipedia.org/wiki/Availability

- MTBF - *Mean Time Between Failures* - predviđeno prosječno vrijeme između dva kvara
  - Za popravljive sustave
- MTTF - *Mean Time To Failure* - predviđeno prosječno vrijeme do kvara
  - Za nepopravljive sustave
- MTTR - *Mean Time To Recovery* - predviđeno prosječno vrijeme potrebno da se otkrije i popravi pogreška

A = MTBF / (MTBF + MTTF) → **0 ?**

3. Dostupnost (*Availability*) (3)

- Koja nam je ciljana dostupnost? 99.9% +
- https://sre.google/sre-book/availability-table/#tablea1

	per year	per quarter	per month	per week	per day	per hour
90%	36.5 days	9 days	3 days	16.8 hours	2.4 hours	6 minutes
95%	18.25 days	4.5 days	1.5 days	8.4 hours	1.2 hours	3 minutes
99%	3.65 days	21.6 hours	7.2 hours	1.68 hours	14.4 minutes	36 seconds
99.5%	1.83 days	10.8 hours	3.6 hours	50.4 minutes	7.20 minutes	18 seconds
99.9%	8.76 hours	2.16 hours	43.2 minutes	10.1 minutes	1.44 minutes	3.6 seconds
99.95%	4.38 hours	1.08 hours	21.6 minutes	5.04 minutes	43.2 seconds	1.8 seconds
99.99%	52.6 minutes	12.96 minutes	4.32 minutes	60.5 seconds	8.64 seconds	0.36 seconds
99.999%	5.26 minutes	1.30 minutes	25.9 seconds	6.05 seconds	0.87 seconds	0.04 seconds

"Three nines"

"Four nines"

"Five nines"

Detaljnija tablica: https://en.wikipedia.org/wiki/High_availability

Kako postići visoku dostupnost (*High Availability*)?

- -> Otpornošću na pogreške (***Fault tolerance***)
- Otpornost na pogreške je svojstvo koje omogućuje sustavu da nastavi ispravno raditi i biti dostupno u slučaju kvara jedne ili više njegovih komponenti.

https://en.wikipedia.org/wiki/Fault_tolerance

- Koje pogreške?
  - Ljudska pogreška (vidi post mortems)
  - Softverske greške
  - Hardverske greške

Kako postići HA/FT?

- **Tri načela**:

1. Eliminiranje SPoF :
   - Replikacija
   - Redundancija
   - Potrebno je imati "*reliable crossover*", pouzdan rezervni sustav jer on može postati SPOF
2. Uočavanje pogrešaka (i izolacija pokvarenih komponenti)
   - Sustav za nadgledanje (npr. Prometheus)
     - False positives (ok) i false negatives (!ok)
3. Oporavak od pogreške:
   - Izbacivanje komponente
   - Ponovo pokretanje komponente
   - Rollback na zadnje ispravno stanje

SLA, SLO, SLI

- ***Service Level Agreement*** je ugovor (pravni dokument) između jednog ili više pružatelja usluga i jednog primatelja usluge koji definira razne aspekte usluge - kvalitetu usluge, dostupnost, dužnosti, itd.
  - Sadrži N SLO-a
  - Definira i (financijske) kazne
- ***Service Level Objective*** definira ciljanu razinu nekog atributa kvalitete, npr.:
  - Usluga će biti dostupna 99.9% vremena u razdoblju od jedne godine
  - Vrijeme odziva će u 90% slučajeva biti manje od 1.5s
  - …
- ***Service Level Indicator*** - mjera usklađenosti s SLO-ima
  - Mjerimo post-festum

https://en.wikipedia.org/wiki/Service-level_agreement

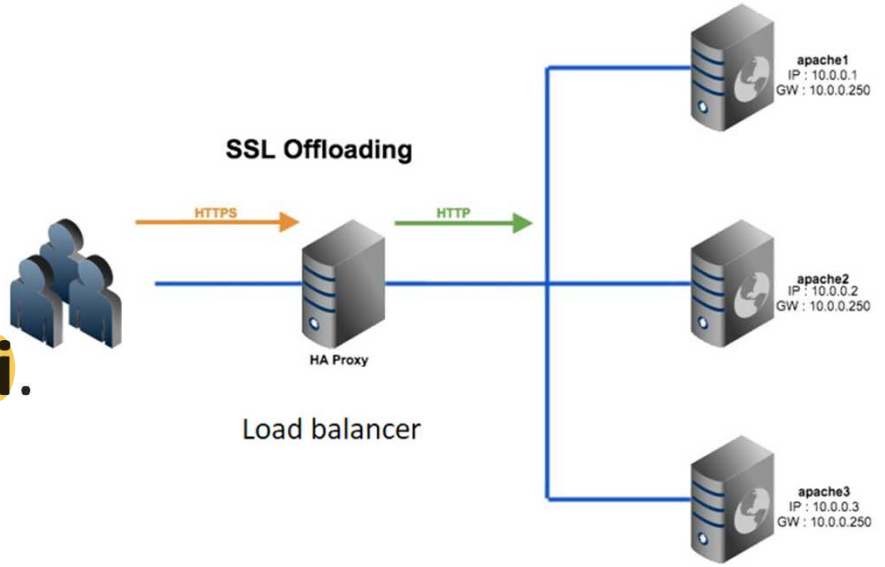
Ali kako?

Architecture building blocks, patterns & techniques

U nastavku ćemo pogledati neke tehnike, obrasce i gradivne blokove koji nam omogućuju da ostvarimo ciljane razine atributa kvalitete, ponajprije **skalabilnosti**

Load balancing (uravnoteženje opterećenja) (1)

- **Uravnoteženje opterećenja**: proces u kojem **raspodjeljujemo zadaće** na **skup radnika** kako bi cjelokupni proces bio **učinkovitiji**.



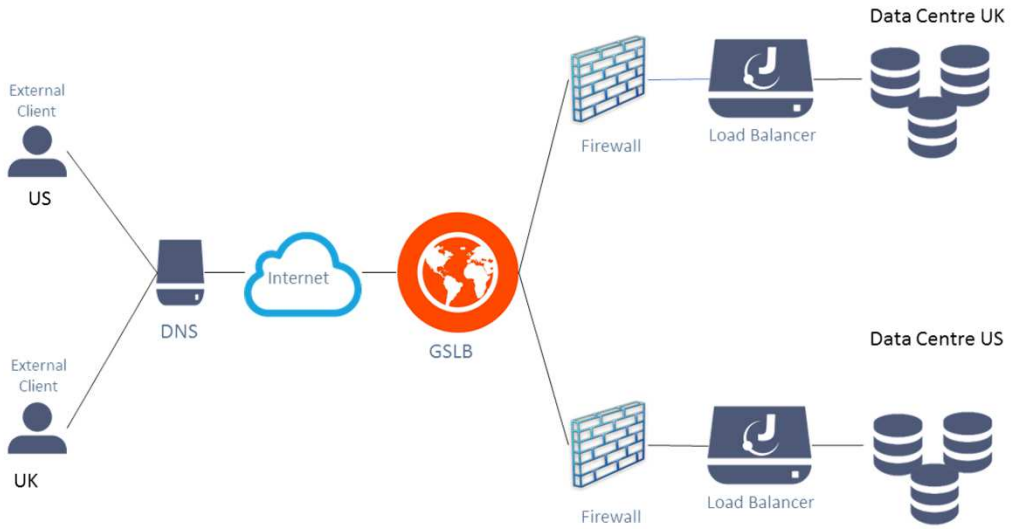
- *Single server abstraction*
- Prednosti:
  - *High **scalability**, auto-scaling (u oblaku)*
  - **HA** - ugasi server koji ne radi
  - Učinkovitost
    - Iako, LB dodaje malo latencije
  - ***Maintenability***, možemo ugasiti server, raditi na njemu

Load balancing (uravnoteženje opterećenja) (2)

- Tipovi:

1. **DNS LB-ovi**
   - Prednost: jednostavan, robustan, brz
   - Nedostatci: ne prati radi li server, podatci mogu biti keširani/zastarjeli, strategija je uvijek jednostavna - *round robin*
2. **Hardverski i softverski LB-ovi**
   - Mogu uzeti u obzir stanje server, snagu, trenutno opterećenje, itd.
   - Može biti i mreža (hijerarhija) internih LB-ova
3. **GSLB: *Global Server Load Balancer*** = 1 + 2

   1. Ipak, pati od keširanja



https://www.edgenexus.io/blog/mr-worldwide-presents-brief-overview-global-server-load-balancing-gslb/

LB softver - informativno

- **HAProxy** *is a free, very fast and reliable reverse-proxy offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for very high traffic web sites and powers a significant portion of the world's most visited ones. Over the years it has become the de-facto standard opensource load balancer, is now shipped with most mainstream Linux distributions, and is often deployed by default in cloud platforms*.

- **NGINX** - *Load balance HTTP traffic across web or application server groups, with several algorithms and advanced features like slow-start and session persistence*
  - https://www.nginx.com/blog/nginx-load-balance-deployment-models/
  - https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/

- Cloud based LBs & GSLB rješenja:

  - Amazon: Amazon ELB, Amazon Route 53
  - Microsoft Azure Load Balancer & Azure Traffic Manager
  - GCP Cloud Load Balancer & Cloud DNS
  - …

Content Delivery Networks

- CDN je **geografski distribuirana** mreža *proxy* servera i njihovih podatkovnih centara. Cilj je pružiti **visoku dostupnost** i **performanse** prostornom distribucijom usluge u odnosu na krajnje korisnike.

https://en.wikipedia.org/wiki/Content_delivery_network

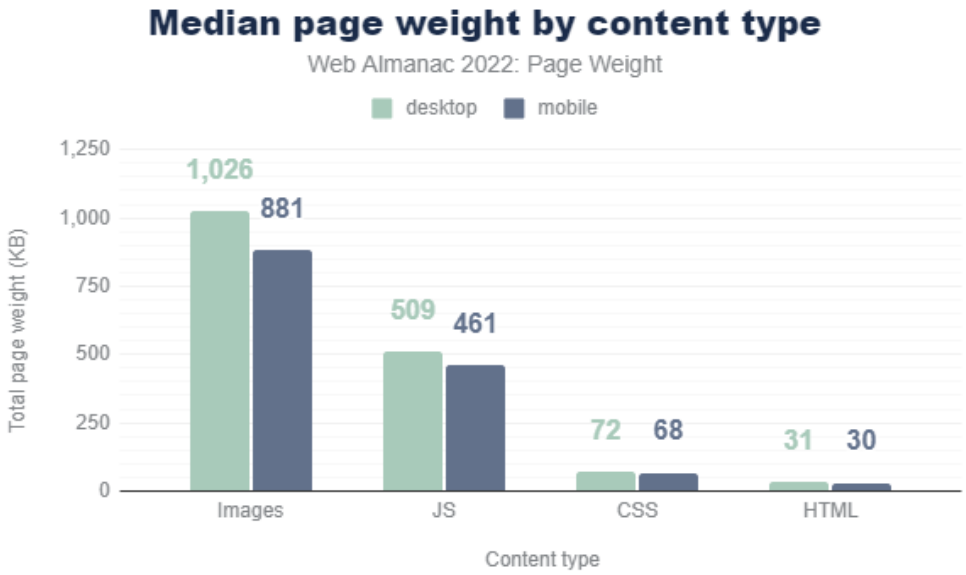
- Strateški raspoređeni, fizički bliži korisnicima
- Keširaju: slike, video, kôd (JS, CSS, …)
  - Još dosta HTTP1.1: https://w3techs.com/technologies/history_overview/site_element/all
- Kao klijenti CDN-a, imamo dvije strategije publiciranja:
  1. ***Pull*** – CDN povlači od nas resurse i kešira ih, osvježava kad isteknu
  2. ***Push*** – sami publiciramo (*push*) na CDN promjene
     - Može se izvesti kao dugi TTL i ručno brisanje keša na CDN-u

Higijena: kako (besplatno) smanjiti bandwidth?

- https://almanac.httparchive.org/en/2022/page-weight

594%

Figure 21.7. Growth in mobile page weight over 10 years

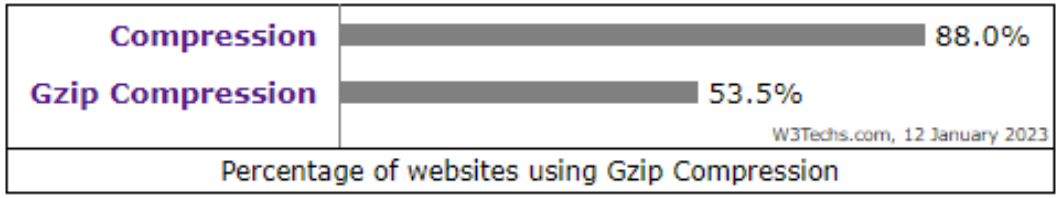


Show description of Figure 21.8

Figure 21.8. Median page weight by content type.

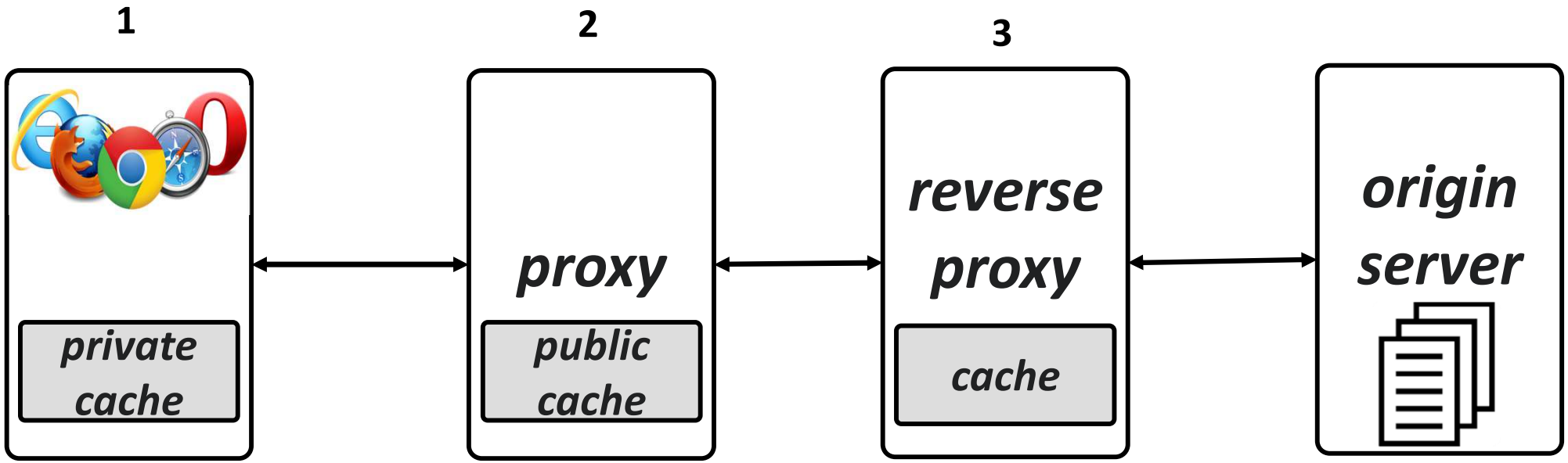
Higijena: kako (besplatno) smanjiti bandwidth?

- **Kôd**: smanjiti (*minify*) JS i CSS, smanjiti broj datoteka (HTTP zahtjeva)
- **Slike**: odabrati pravi format, više rezolucija (<picture>), *lossy/lossless*, alati za optimizaciju slika
  - Kad png, kad jpg, koja je razlika?
- **Za sve** što može: offload na CDN
- HTTP:
  - Content-Encoding: gzip
    - Lagano – „uključiti”
    - Ne koristiti za slike i video, tj. već komprimirane formate
  - Cache
    - Što keširati? Ne: HTML, Da: slike, video, JS+CSS, …
    - → Sljedeći slajd



HTTP cache

- *Cache* se općenito može nalaziti **na tri mjesta**:
  1. Na pregledniku (private)
  2. (ako postoji) Na N proxy/CDN poslužitelja (public)
  3. (ako postoji) Na „našem” *reverse proxy* poslužitelju
     - Često ujedno i LB
     - Npr. Varnish ili NGINX



HTTP Cache Headers

Klijent- Request (Validacija)

- `If-None-Match` ⟷ `ETag`
- `If-Modified-Since` ⟷ `Last-Modified`

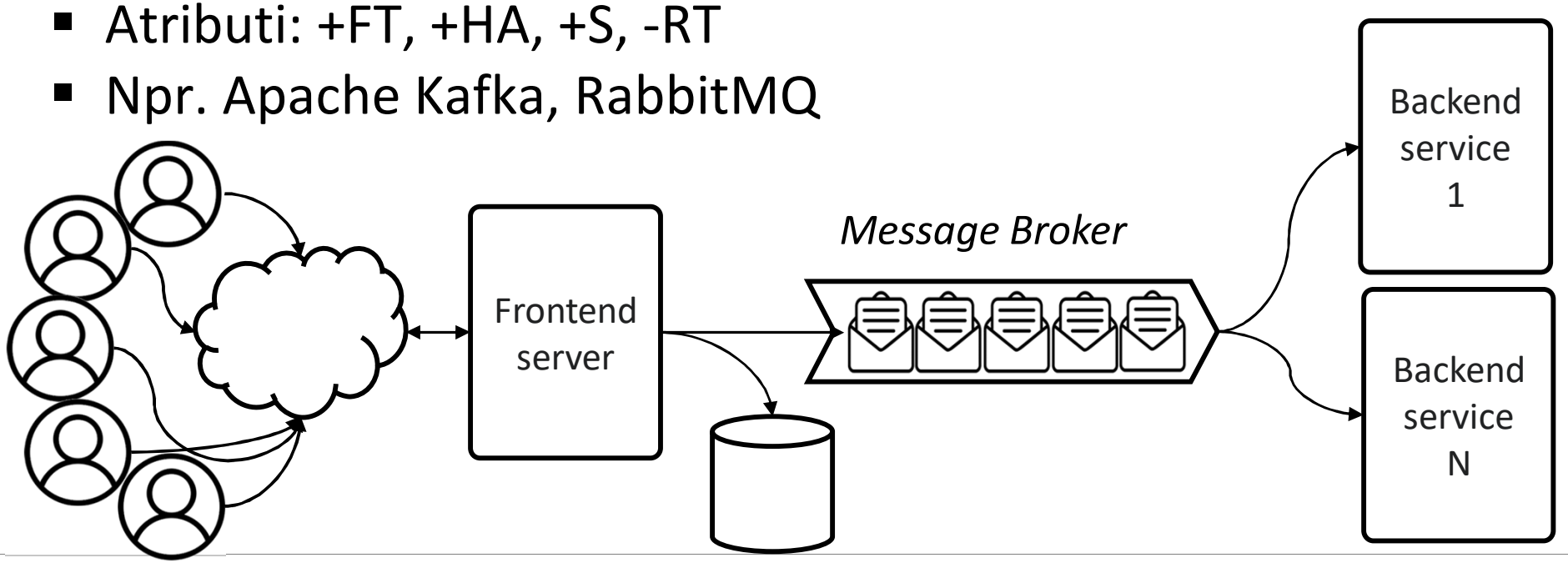
Server - Response

- `expires`
- `pragma`
- `Cache-Control: <directive> [, <directive>]*`
- **Directives**:
- **Public**: This directive indicates that the response can be stored by any cache without any restriction. In case the response is non-cacheable, it can still be cached.
- **Private**: It indicates that only browser cache is eligible to store the response.
- **no-cache**: It indicates that the response can be stored by any cache without any restriction even if it is non-cacheable. The condition that needs to be satisfied here is that the stored response must be validated by the origin server before being used.
- **no-store**: It indicates that the response cannot be stored by any cache.
- **max-age=<seconds>**: It indicates the maximum amount of time a resource remains fresh and can be requested to be accessed.
- …

https://www.geeksforgeeks.org/http-headers-cache-control/

Building block: Message Broker

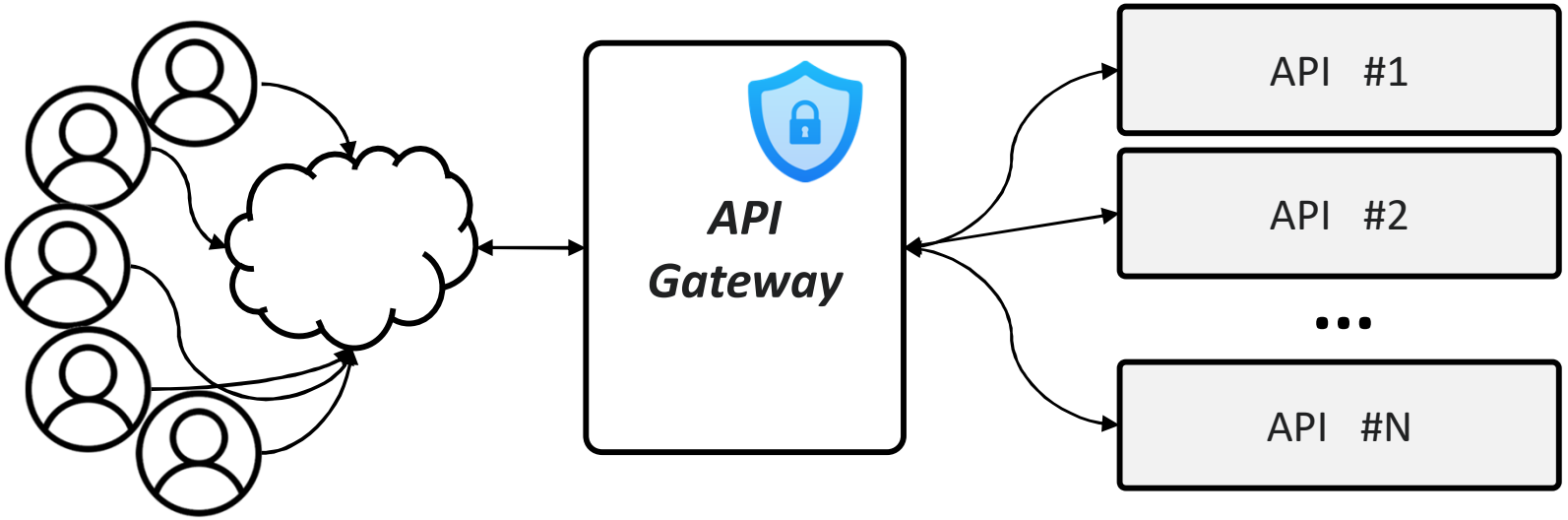
- **Asinkrona** komunikacija, kada je zahtjev **zahtjevan** ☺
- *Message Broker* (posrednik poruka):
  - Pohranjuje poruke između pošiljatelja i primatelja u **red (*queue*)**
  - **Interni mehanizam**, može ih biti N u našem sustavu
  - Uklanja spregu (*coupling*) između pošiljatelja i primatelja
    - Dodatno, može: pohranjivati poruke, transformirati, LB, …
    - Tipično omogućuju pub/sub
  - Atributi: +FT, +HA, +S, -RT
  - Npr. Apache Kafka, RabbitMQ



API Gateway (1)

- Kod rasta sustava često dolazi do razdvajanja API-ja u N tematski/logički grupiranih API-ja (servisa)
- ***API Gateway*** je alat za upravljanje API-jem koji se nalazi između klijenta i skupa pozadinskih usluga. Radi kao *reverse proxy* za prihvaćanje svih API poziva, agregiranje različitih usluga potrebnih za njihovo obavljanje i vraćanje odgovarajućeg rezultata.

https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do



API Gateway (2)

- Prednosti*:
  - Lakše interne izmjene
  - Sigurnost (autentifikacija, autorizacija) na jednom mjestu
  - Keširanje statičkog sadržaja
  - Nadziranje i upozoravanje (*monitoring and alerting*)
  - Protocol translation: npr. JSON – gRPC, HTTP-HTTPS, JSON-legacy XML, itd.
- Dobre prakse:
  - Ne smije sadržavati poslovnu logiku!
  - Može postati SPOF, ali možemo ih replicirati + LB
  - Ne zaobilaziti Gateway!
- Dodaje lagani RT overhead

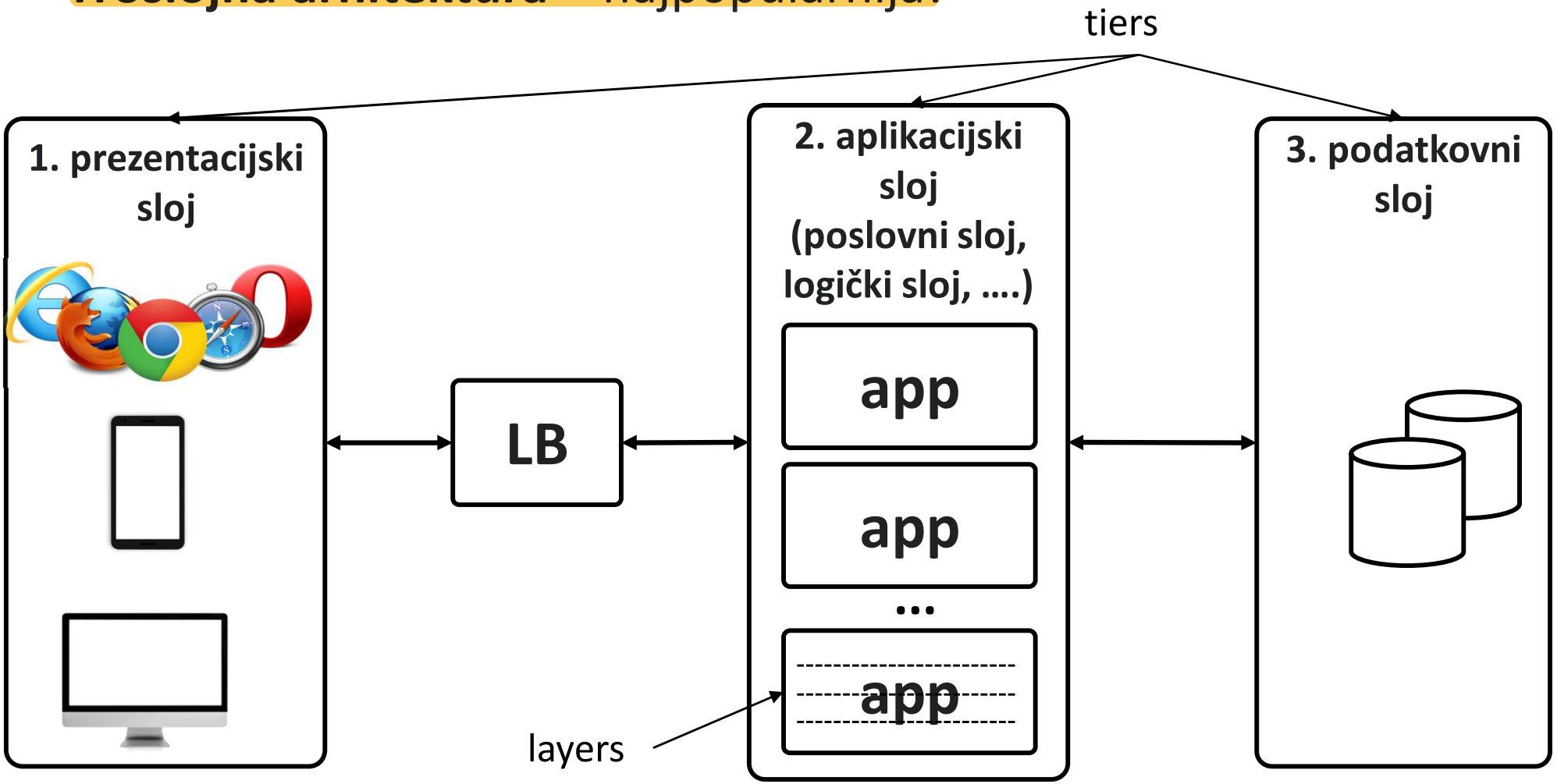
* Software Architecture & Design of Modern Large Scale Systems, Michael Pogrebinsky

Arhitekture web-aplikacija

… i još neki obrasci…

Multi-tier (višeslojna) arhitektura

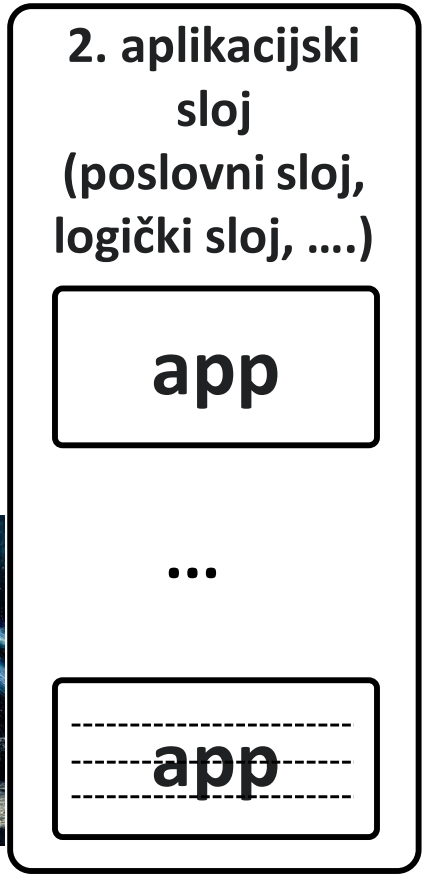
- Neprecizna terminologija na hr i eng:
  - ***Tier*** (fizička separacija) <> ***Layer*** (logička separacija softvera)
  - Nije nužno 1:1
- **Troslojna arhitektura** – najpopularnija!



Troslojna arhitektura

- ✓ Dobra praksa - poslovna logika u aplikacijskom sloju
- ✓ Razvojni timovi razdvojeno raditi na slojevima (*tiers*)
- ✓ Možemo elegantno **skalirati** aplikacijski sloj (i ostale)
  - x Ali - skaliramo "sve ili ništa" ☹
- ✓ Dobra početna općenamjenska arhitektura

- x Temeljni nedostatak je **monolitna** struktura aplikacija u **aplikacijskom sloju**
- x **Prevelike cjeline**, „previše kôda”:
  - Teža podjela poslova među razvojnim timovima - organizacijska skalabilnost
  - Teže održavanje
  - Teže/sporije publiciranje novih verzija
    - „*overkill*” – sitna izmjena znači publiciranje kompletne aplikacije
    - što ako uvedemo pogrešku?



Varijante

- 2T, 4T, …
- Izdvajanje servisa, API, API Gateway
- -> modularizacija

Mikroservisna arhitektura

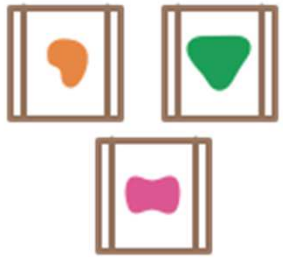
- Mikroservisna arhitektura strukturira aplikaciju kao skup usluga/servisa koje su:
  - Samostalno rasporedive (*indenedently deploy*)
  - Labavo spregnute (*loosely coupled*)
  - Organizirane oko poslovnih mogućnosti
  - U vlasništvu malog tima
  - Vrlo lako se održavaju i testiraju
  - Arhitektura mikroservisa omogućuje brzu, čestu i pouzdanu isporuku velikih, složenih aplikacija. Također omogućuje organizaciji da evoluira tehnologiju

https://microservices.io/

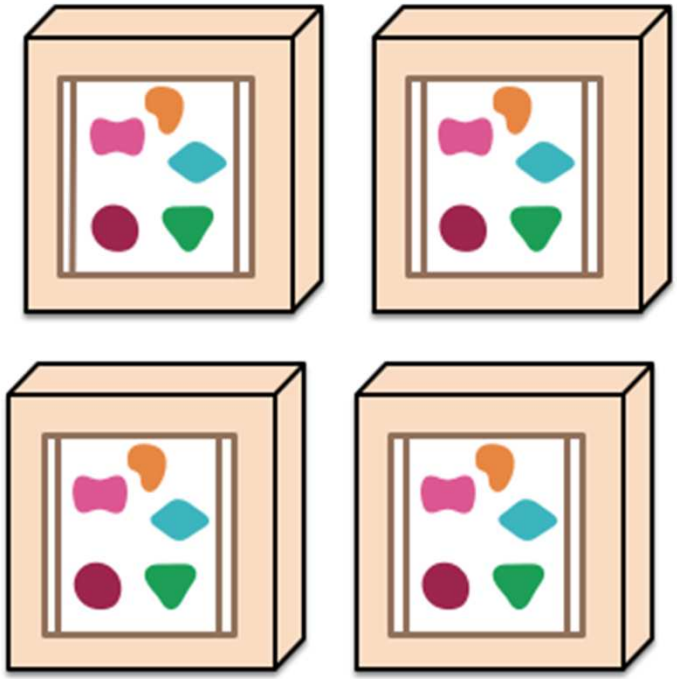
Mikroservisna arhitektura

A monolithic application puts all its functionality into a single process...

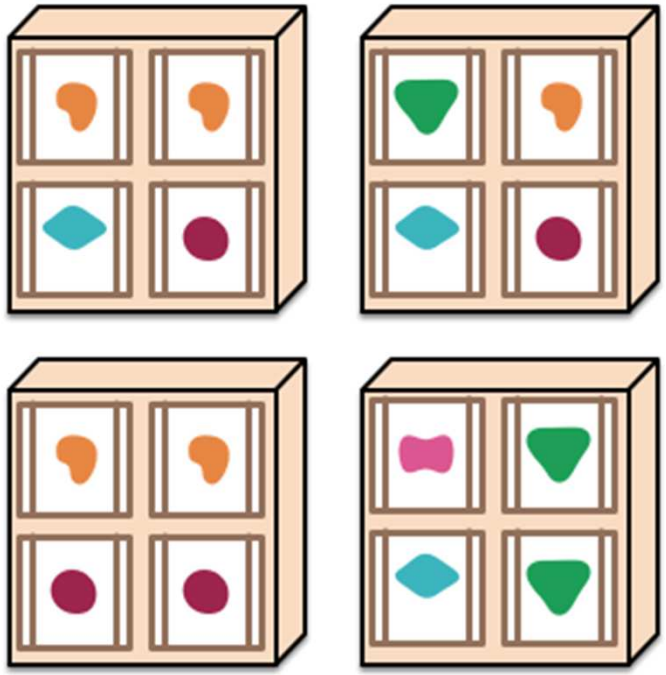
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers

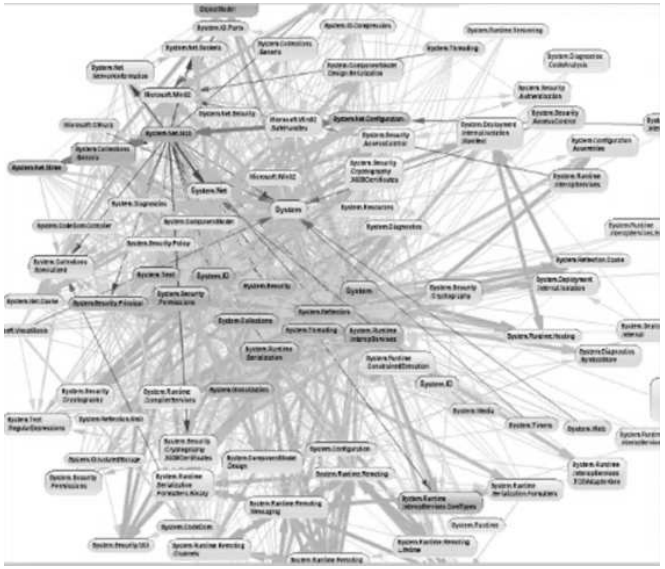


... and scales by distributing these services across servers, replicating as needed.



https://martinfowler.com/articles/microservices.html

Mikroservisna arhitektura (3)



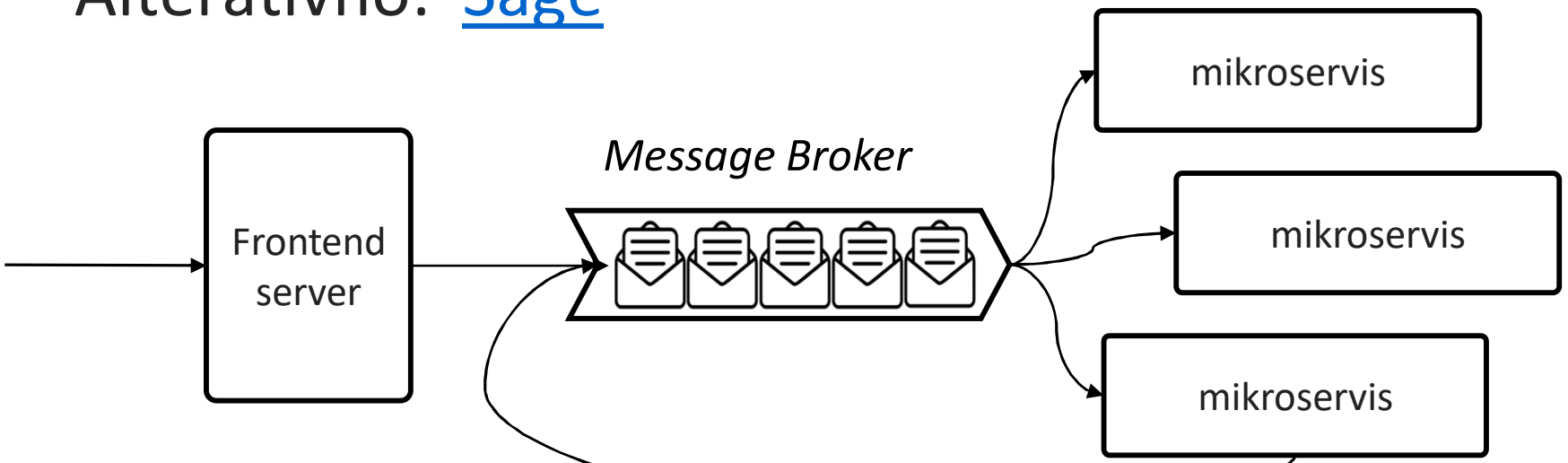
- Niz prednosti:
  - ✓ Skalabilnost
  - ✓ Organizacijska skalabilnost
  - ✓ Manji kod
  - ✓ FT - lakša izolacija pogrešaka
- Ali i mnogobrojni izazovi:
  - Nije lako ustrojiti i odrediti granice servisa
    - Single Responsibility Principle– there should not be multiple responsibilities in a microservice.
  - Database per service
  - *Big ball of mud*
  - ...

Sam Newman, author of *Building Microservices* and *Monolith to Microservices*, told attendees at the QCon developer conference in London that "microservices should not be the default choice…. **but a last resort**"

https://www.theregister.com/2020/03/04/microservices_last_resort/

Event driven arhitektura (temeljena na događajima)

- Događaj (*event*) - maleni, samostalni, nepromjenjivi zapis nečega što se dogodilo
  - Sadrži opis događaja i trenutak događaja (event time)
  - Npr. dodavanje proizvoda u košaricu
- Omogućuje nam da smanjimo međusobnu ovisnost (mikro)servisa
- Naslanja se na *Message Brokere*
- Alterativno: Sage



Event sourcing

- Alternativni način pohrane podataka – svaka **promjena** se pohranjuje kao **nepromjenjivi zapis – *event***.
  - *Model information about activity in the domain as a series of discrete events. Represent each event as domain object.*

    Eric Evans, Domain-Driven Design Reference
- Događaji se pohranjuju kao *append-only* niz, izmjena ili brisanje je zabranjeno ili se u najmanju ruku obeshrabruje
- Točnije: klijenti izdaju naredbe (***commands***) koje, ako uspješno prođu validaciju, postaju nepromjenjivi događaji – ***events***
  - Npr. kupnja karte, ako ima slobodnog mjesta, postaje event

- Prednosti: fleksibilno, omogućuje temporalne upite, objašnjava što se dogodilo (klasični pristup samo ažurira), reproducibilno, itd.
- Nedostatci: složenije, sporije, tipično u paru s CQRS obrascem i materijaliziranom read-only bazom, zahtjeva pouzdanu mrežnu infrastrukturu radi osjetljivosti na vrijeme, heterogeni eventi

CQRS - *Command Query Responsibility Segregation*

- Razdvaja se čitanje i pisanje - dvije različite baze podataka:
  - Baza u koju pišemo propagira događaje pisanja
  - Koji se konsolidiraju u read-only bazu za čitanje

✓ Poboljšava skaliranje kad imamo **istovremeno visoke zahtjeve na čitanje i pisanje**

- Može se (i bolje) primijeniti samo na dio sustava

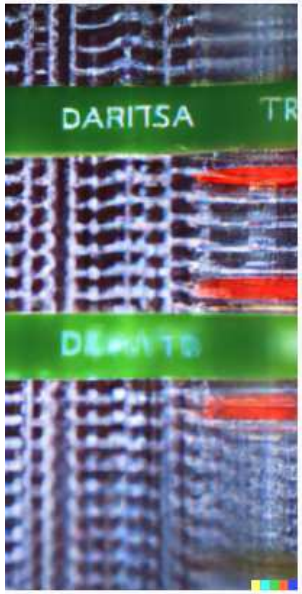
x Donosi značajnu dodatnu složenost

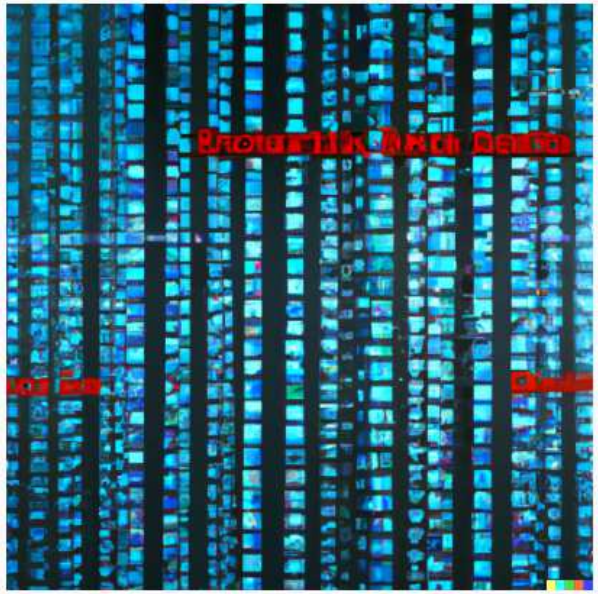
> *Despite these benefits, **you should be very cautious about using CQRS**. Many information systems fit well with the notion of an information base that is updated in the same way that it's read, adding CQRS to such a system can add significant complexity.*
>
> Martin Fowler

Serverless architecture

- model u kojem **treći pružatelj (*3rd party*)** usluga (tipično u oblaku) sam brine o poslužitelju.
- Programeri moraju brinuti o kodu, a sve ostalo radi pružatelj poslužitelja:
  - ✓ sigurnosne zakrpe,
  - ✓ balansiranje opterećenja,
  - ✓ upravljanje kapacitetom,
  - ✓ skaliranje,
  - ✓ bilježenje i praćenje.
- Cijela ili dio aplikaciju na modelu bez poslužitelja
- "Prijatelji":
  - *Microservices* - **logička** organizacija/separacija koda
  - *Serverless* - **fizička**, način deploymenta, obavljanja

Podatkovni sloj





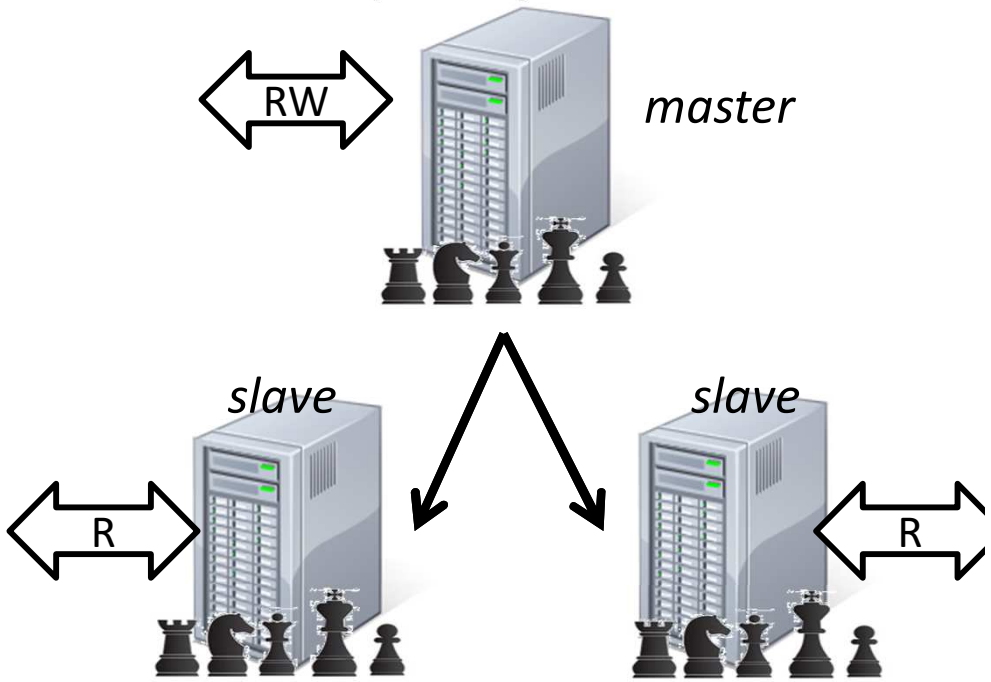
Podatkovni sloj

- Strategija skalabilnosti usmjerena na podatkovni sloj, tj. bazu podataka web aplikacije
- Dva glavna razloga za skaliranje baze podataka:
  1. prevelik broj pristupa bazi
     → replikacija
  2. prevelika količina podataka koje treba pohraniti
     → fragmentacija (*sharding, partitioning*)

- **Replikacija:** radi se u slučaju da ima puno zahtjeva za čitanje, a malo za pisanje – baza podataka se malo mijenja, a velik broj upita.
- Najčešće **MS**

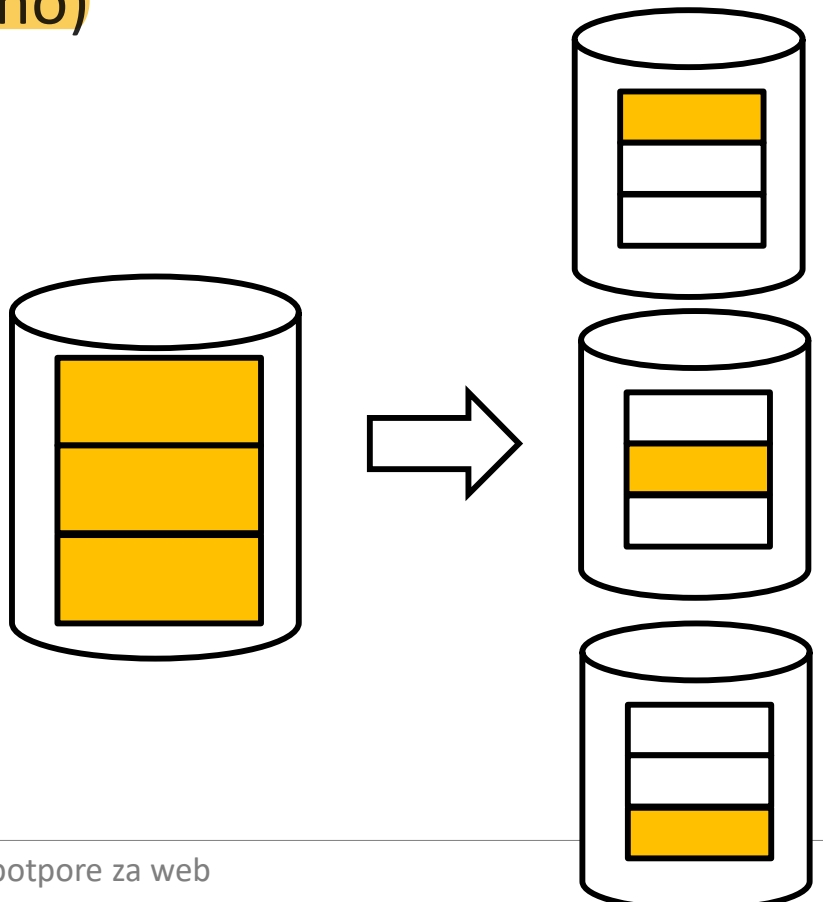
MS replikacija

- Kod **replikacije** podatke iz glavne baze podataka (*master/primary*) repliciramo u pomoćne baze podataka (*slave/secondary*).
- Prilikom svakog izmjene podataka u master, ažuriraju se i slave baze podataka.
- Upiti na baze se sada mogu raspodijeliti na više baza, što smanjuje opterećenje glavne baze. To ne povećava brzinu pisanja, ali glavna baza mora podržati zahtijevanu količinu pisanja.
- Možemo replicirati i replike.
  Problem koji se ovdje pojavljuje je kašnjenje replikacije (***replication lag***) u slučaju da se čita iz replicirane baze prije nego je podatak proslijeđen iz master baze.

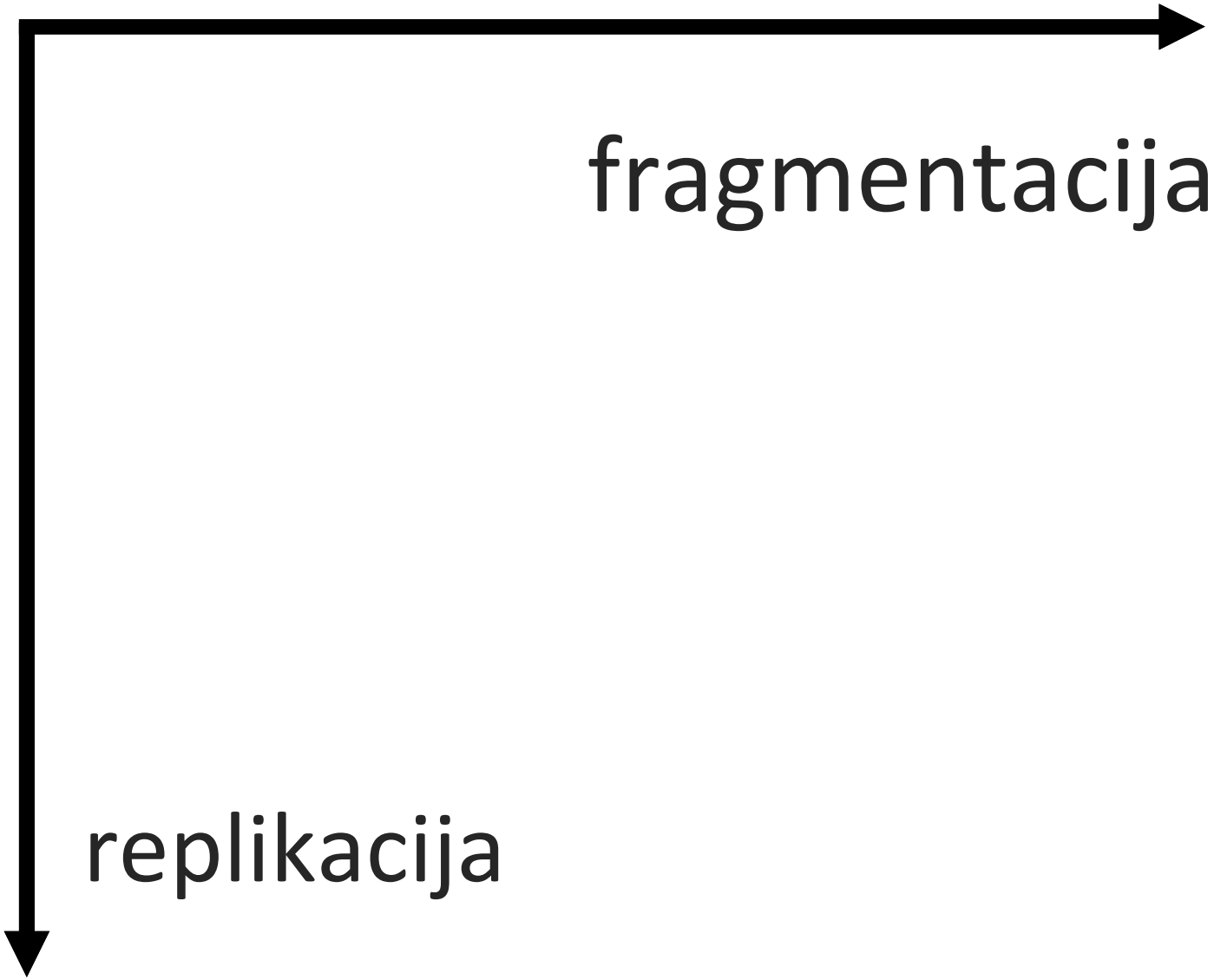


Particioniranje, fragmentiranje – *sharding*

- **Postupak particioniranja (*sharding*):** podjela baze podataka u disjunktni skup fragmenata koji obuhvaćaju sve podatke u bazi podataka uz zadovoljenje pravila da se baza podataka može rekonstruirati iz tih fragmenata bez gubitka informacije
- relacije mogu biti razdijeljene u fragmente horizontalno ili vertikalno (ili horizontalno i vertikalno)
- Više master računala:
- U svaki *master* sprema se po dio podataka, raspodijeljenih po nekom algoritmu (hash, range, …)
- Dodatna kompleksnost, usmjeravanje upita
- Slabije podržano u relacijskim sustavima, nije lako zadržati ACID svojstva



Fragmentacija & Replikacija su ortogonalni



NoSQL sustavi

- NoSQL je široki pojam koji se koristi za označavanje raznih tipova pohrane podataka koji se razlikuju od tradicionalnog modela relacijskih baza
- NoSQL baze su više specijalizirane i pogodnije za skaliranje od tradicionalnih SQL baza
- Modeli podataka: *key-value, document, graph, column-family*
- Tipično podržavaju i replikaciju i fragmentaciju (izuzev graph)

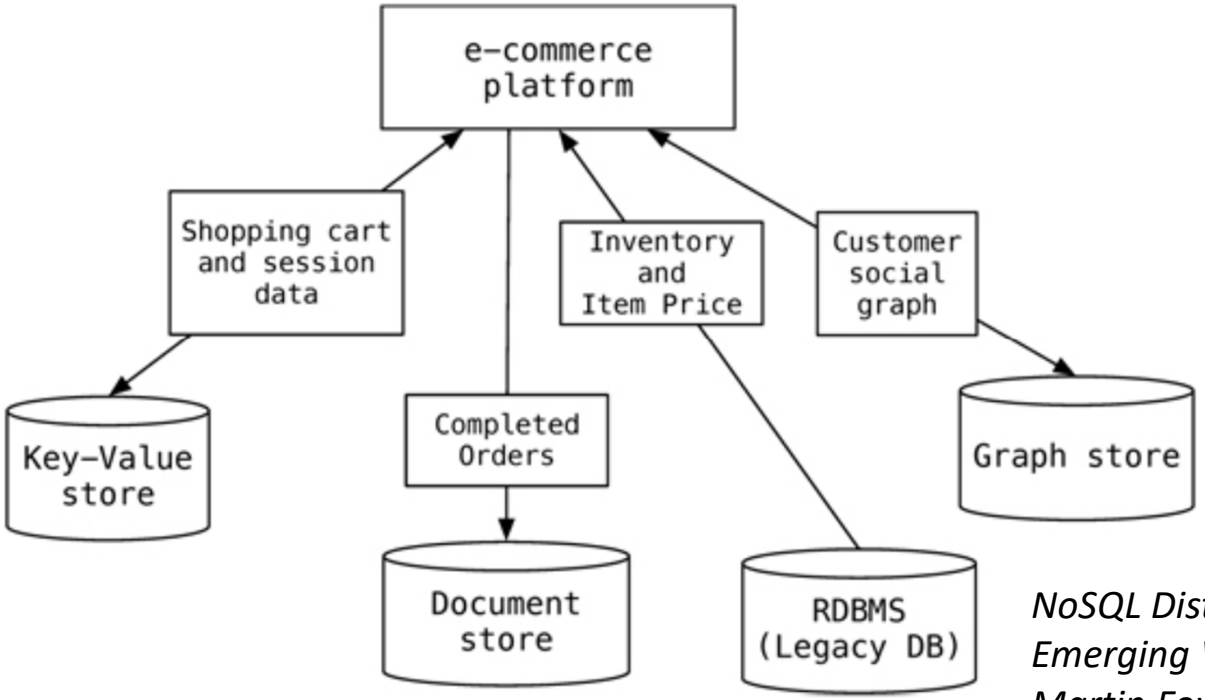
NoSQL vs RBP

RDB su najbolje
one-size-fits-all
rješenje

NoSQL su **specijalizirana**
rješenja za određene
(grupe) problema

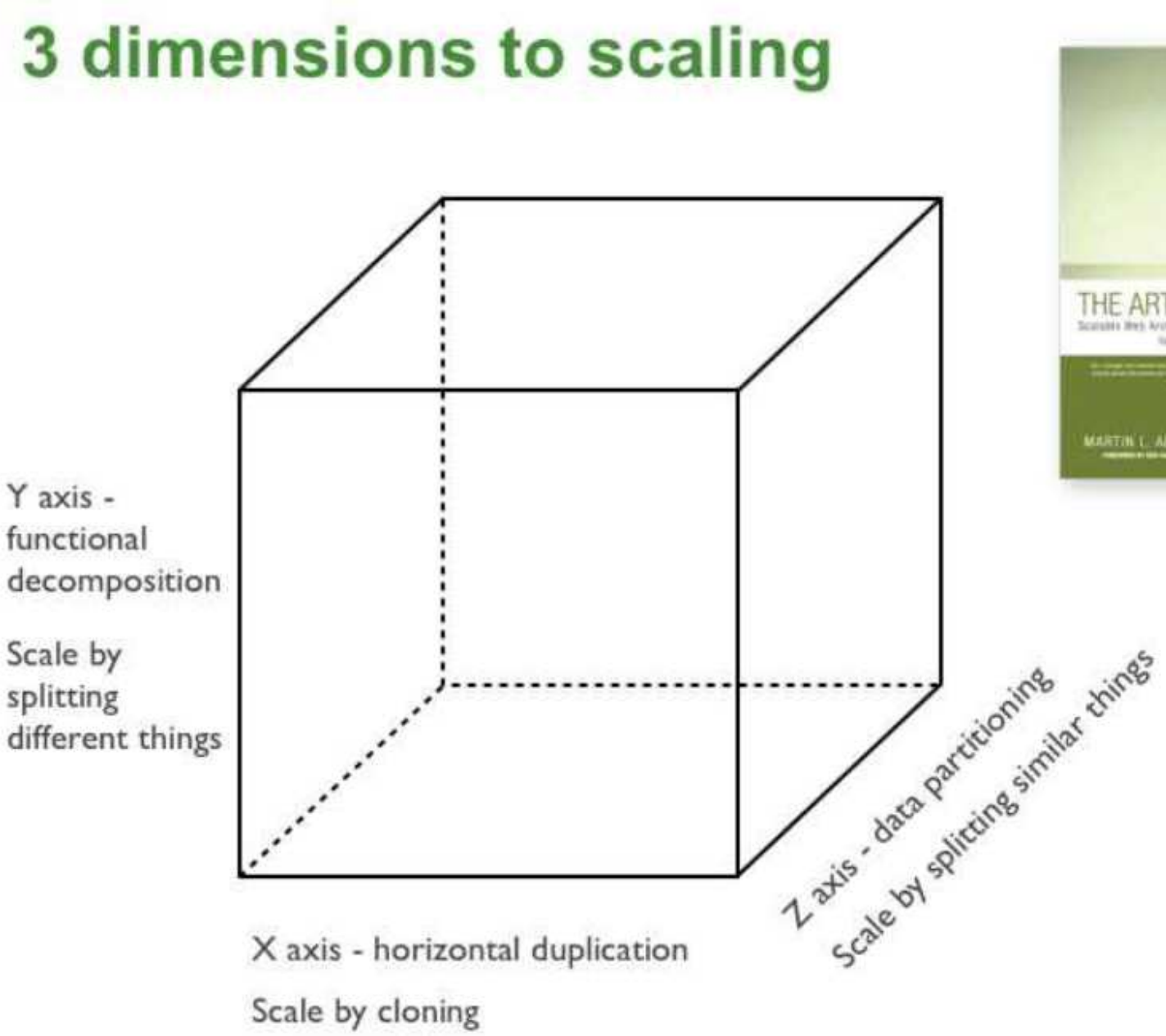
Polyglot persistance (1)

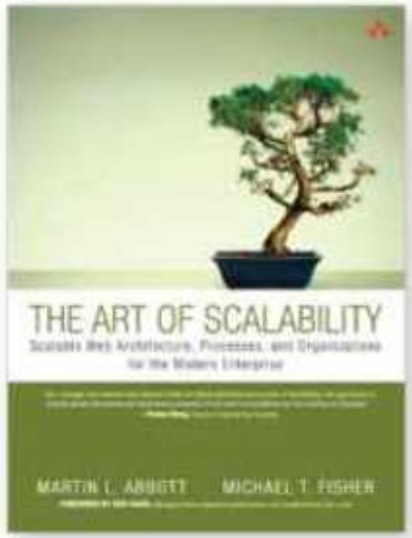
- Različite baze podataka su namjenjene za rješavanje različitih problema
- Hibridni pristup perzistenciji podataka: kombiniranje tehnologija kako bi se najbolje zadovoljile različite klase potreba za perzistencijom podataka
- Npr.



NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Martin Fowler

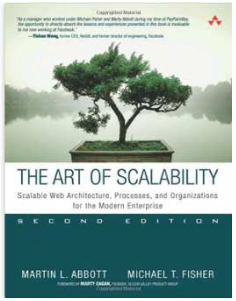
The Scale Cube - tri dimenzije skaliranja



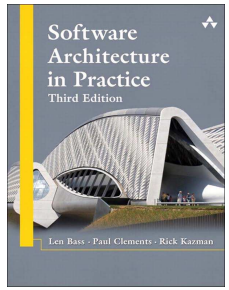


Literatura

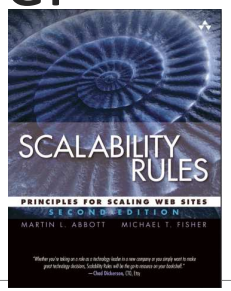
- Software Architecture & Design of Modern Large Scale Systems, Michael Pogrebinsky
- The Art Of Scalability, by Martin Abbott (Author), Michael Fisher (Author)

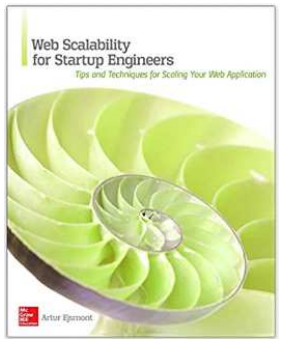


- Software Architecture in Practice (SEI Series in Software Engineering) by Len Bass , Paul Clements, et al
- Web Scalability for Startup Engineers, Tips & Techniques for Scaling Your Web Application, Artur Ejsmont



- Scalability Rules, Principles for Scaling Web Sites, Martin L. Abbott, Michael T. Fisher
- + reference u dokumentu





Za one koji žele znati više…

… i dobiti par bodova više…

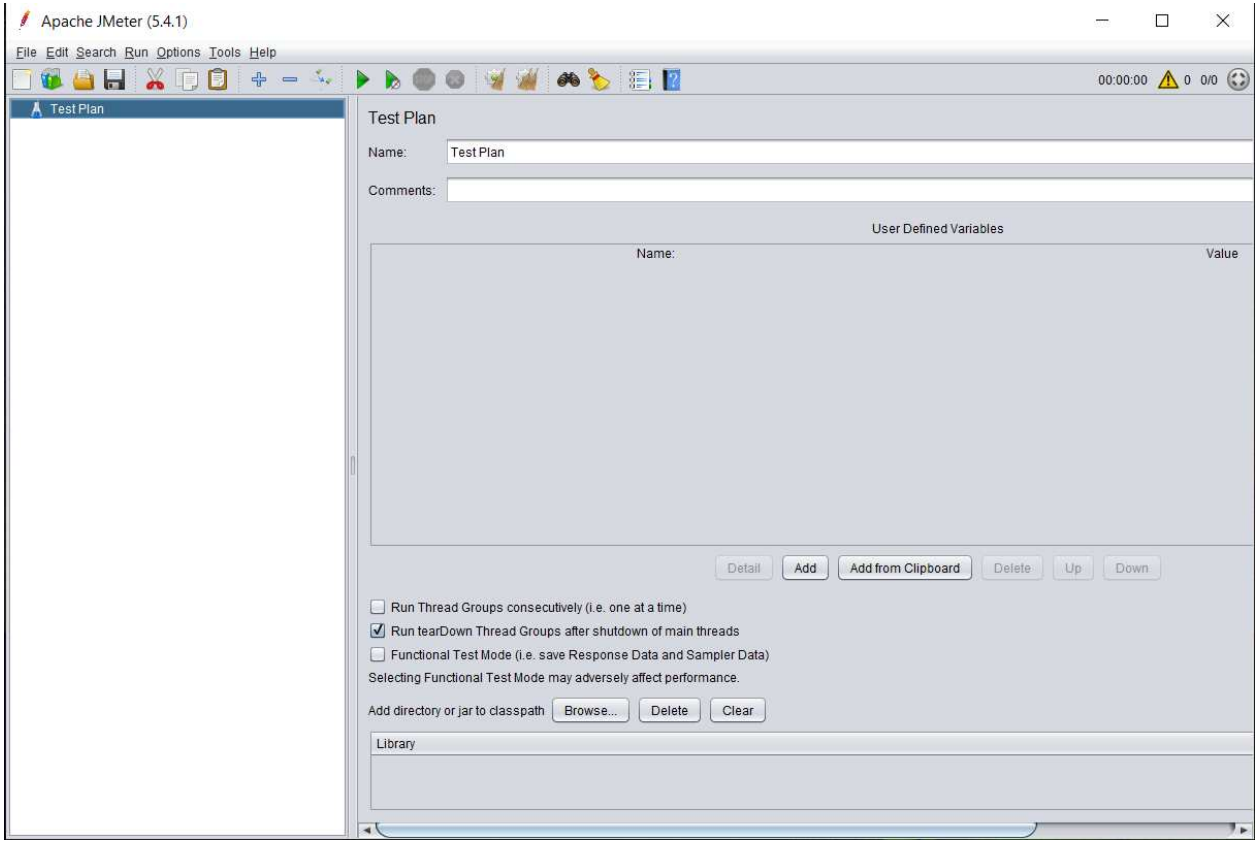
Apache JMeter

JMeter

- Apache JMeter testni alat omogućuje snimanje HTTP zahtjeva prema poslužiteljima i testiranje njihovih performansi
  - Desktop aplikacija otvorenog koda, besplatna, napisana u programskom jeziku Java
  - Provodi testove opterećenja (*load test, stress test*) koji su dio testa prihvaćanja (*acceptance test*) i potrebni prije puštanja aplikacije u produkciju
  - Performanse statističkih resursa (JavaScript i HTML) i dinamičkih (JSP, servleti, AJAX komponente)
  - Pomaže odrediti maksimalni broj istodobnih korisnika web aplikacije
  - Podržava izvještavanje, grafovi i izvješća o učinku (*performance reports*)
  - Preuzimanje: https://jmeter.apache.org/download_jmeter.cgi

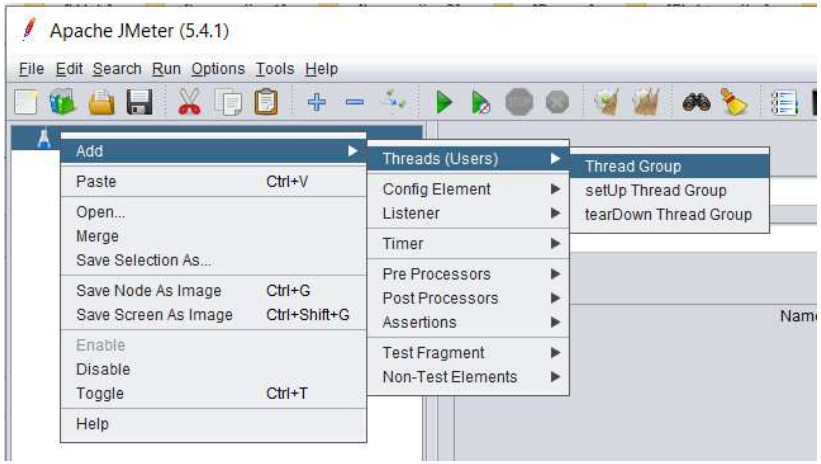
Izrada *Performance Test Plan* (1)

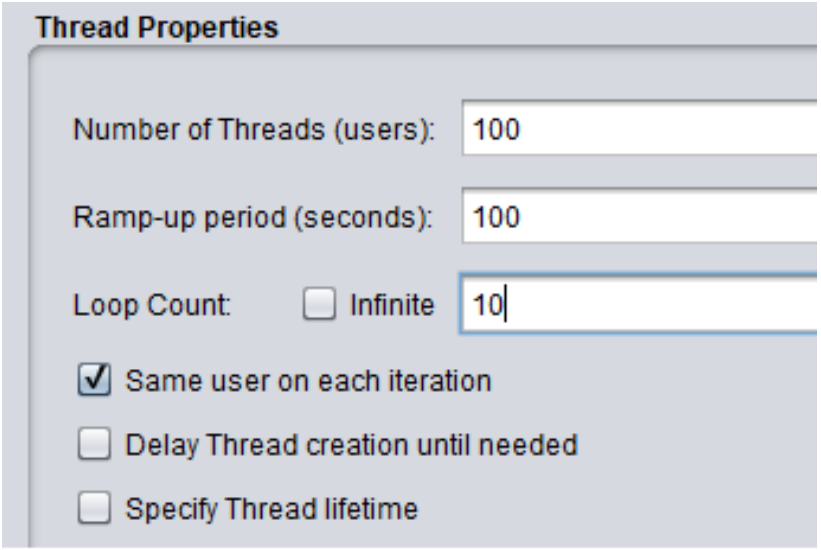
- Postupak izrade *Performance Test Plan*
- Pokrenuti aplikaciju:
  - jmeter.bat
  - ApacheJMeter.jar



Izrada *Performance Test Plan* (2)

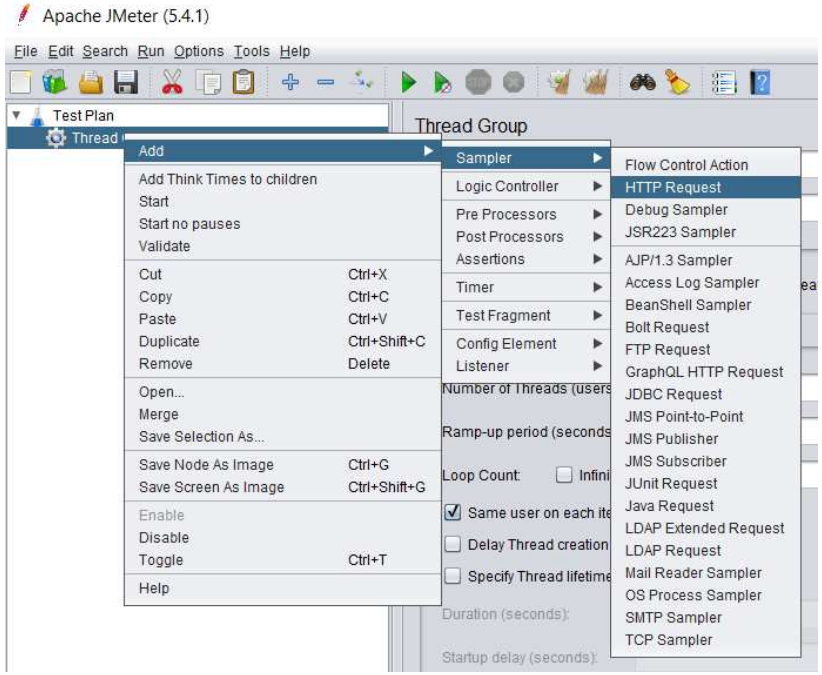
- Odabrati:
  - Add Thread Group
    - Number of Threads: 100
      - Broj simuliranih korisnika koji će povezati s web stranicama
    - Ramp-Up Period: 100
      - Koliko dugo će aplikacija čekati u sekundama prije pokretanja novog simuliranog korisnika. Primjerice, ako imamo 100 korisnika i Ramp-Up period je 100 s, onda će kašnjenje između pokretanja dva uzastopna korisnika biti 1 s (100 s /100 korisnika)
    - Loop Count: 10
      - Koliko puta će se test pokrenuti

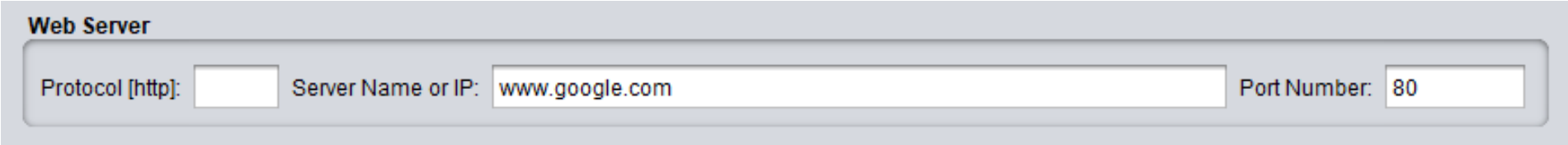




Izrada *Performance Test Plan* (3)

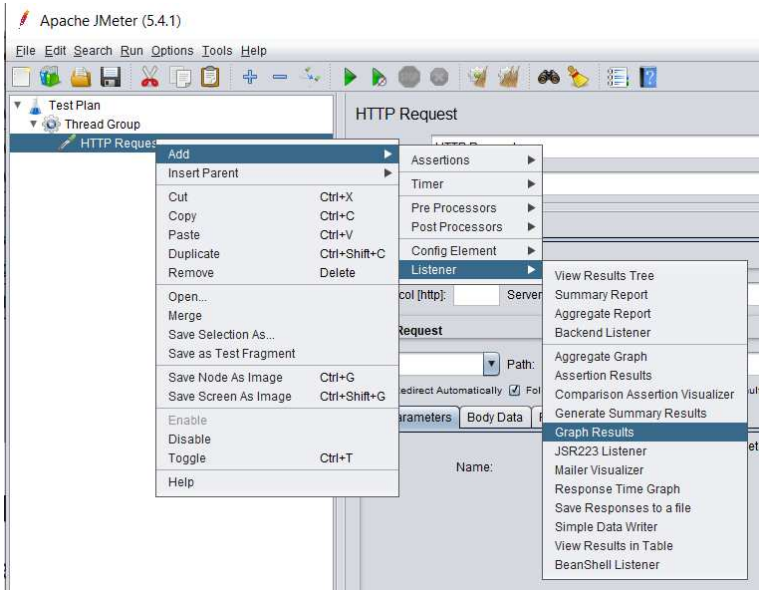
- Dodati JMeter elemente:
  - HTTP Request
    - U polje „Server Name or IP” unesite URL ili IP adresu poslužitelja koji testirate, npr. www.google.com
    - Unesite port na koji se šalju zahtjevi (80)
    - U polje „Path” unesite dodatne elemente zahtjeva koji šaljete na poslužitelja, npr. ako ovdje upišete „calendar” JMeter će izraditi zahtjev „http://www.google.com/calendar” i poslati ga na Google poslužitelj

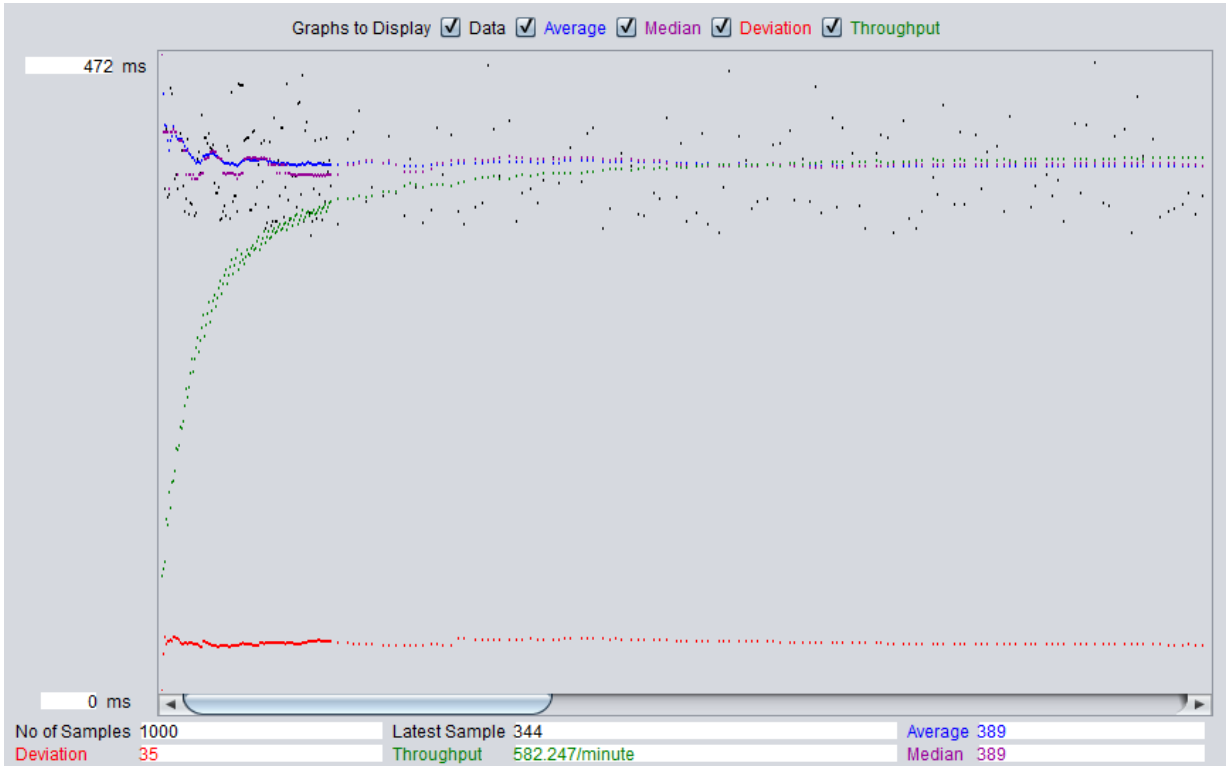




Izrada *Performance Test Plan* (4)

- Dodati izvještaje - vizualizacija rezultata testiranja
  - Odabrati „Graph Results” u meniju „Listener”
  - Podržano je više vrsta izvještavanja
  - Pokrenuti test („Run Test”)  (ili CTRL+R)
  - Rezultati se prikazuju u stvarnom vremenu





Graf prikazuje rezultat plana testiranja u kojemu je simulirano 100 korisnika koji pristupaju web stranicama www.google.com

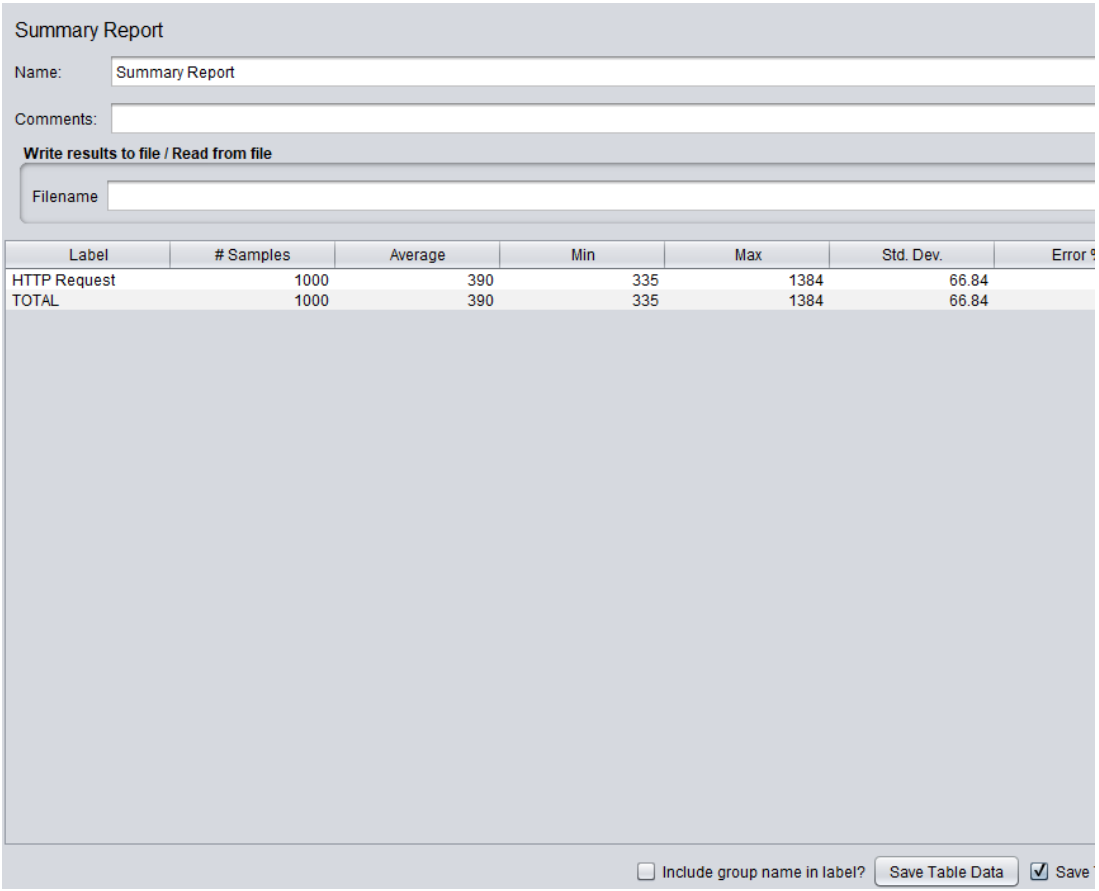
Izrada *Performance Test Plan* (5)

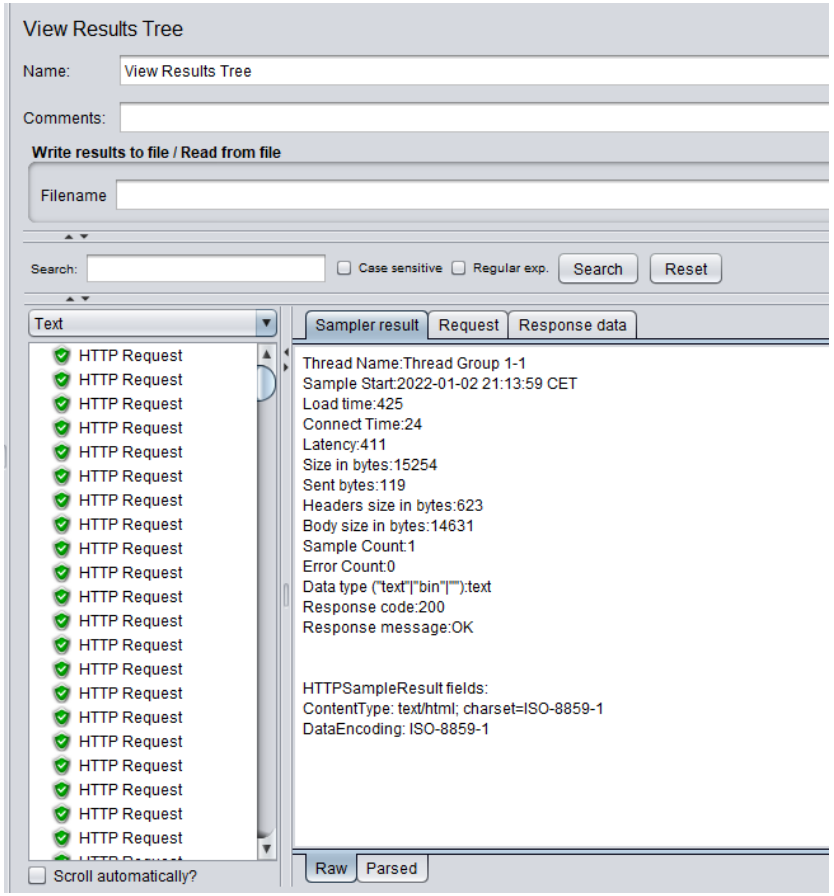
- Analiza rezultata testa
  - Rezultati su predstavljeni u sljedećim bojama:
    - Crna: Ukupan broj poslanih zahtjeva
    - Plava: Trenutni prosjek svih poslanih zahtjeva
    - Crvena: Trenutna standardna devijacija
    - Zelena: Stopa propusnosti (*throughput rate*) koja predstavlja broj svih zahtjeva u minuti koje je poslužitelj obradio
  - Za analizu test opterećenja osobito su važne dvije vrijednosti:
    - Propusnost (*throughput*)
    - Devijacija (*deviation*)
  - Propusnost predstavlja sposobnost poslužitelja da podnosi visoku razinu opterećenja zahtjevima. Viša vrijednost je bolja.
  - U ovom testu propusnost Google poslužitelja bio je 582.247/minute, tj. toliko zahtjeva je poslužitelj mogao obraditi u 1 minuti.
  - Devijacija je u crvenoj boji (35). Manja vrijednost je bolja.

- Isti test može se pokrenuti više puta, rezultati će se agregirati

View Results Tree i *Summary Report*

- Mogućnost View Results Tree
  - Vidljivi su detalji pojedinih HTTP zahtjeva
- Summary Report
  - Završni tablični izvještaj





Ostale aplikacije

- Selenium WebDriver – aplikacija za automatizaciju web preglednika (automatsko upravljanje), W3C preporuka, https://www.selenium.dev/documentation/webdriver/
- Postman – izrada i ispitivanje performansi aplikacijskih programskih sučelja, https://www.postman.com/
- Pact – ispitivanje integriteta HTTP poruka (*contract testing*), npr. za mikroservisne arhitekture, https://docs.pact.io/

API

API are a *contract that we define for other applications* so they ca
use our system without having to know anything about its *interna*
design and implementation

- ~~Ugovor~~
- ~~Tri grupe~~
  - ~~Public, dobra praksa da se registriraju~~
  - ~~Partner~~
  - ~~Private, interno, unutar kompanije~~
  - ~~Good patterns:~~
    - ~~1 complete encapsulation~~
    - ~~2 easy to use, understant~~
    - ~~3. keeping the operations idempotent~~
    - ~~4 api pagination, primjer email~~
    - ~~5 async operations, eg big data report~~
      - ~~Async api - odgovor odmah~~
    - ~~6. versioning api~~

RPC

- ~~Local transparency~~
- ~~From interface: Stubs: server and client~~
- ~~Around for decades…~~
- ~~Pick the framework…~~
- ~~Benefits:~~
  - ~~Convinience, from any lang~~
- ~~Drawback:~~
  - ~~Slow -> introduce async ops for slow methods~~
  - ~~Unreliability Primjer kreditna kartica - idempotentnost~~
- ~~When to use: najčešće server-to-server~~

REST API

- ~~Not a standard or a protocol but style~~
- ~~Vs RPC, REST is resource oriented~~

—

- ~~How to achieve reqs:~~
  - ~~Statelesness~~
  - ~~Cacheability~~
- ~~Named resources~~
  - ~~Simple, singular~~
  - ~~Collection, plural names~~
  - ~~Best practice: nouns only, meaningful names, ids should be uniique and url friendly~~